Test and Training ENabling

Architecture (TENA)

TENA BASELINE PROJECT REPORT

Volume IV
Technical Reference Architecture

Review Copy

Under Secretary of Defense (Acquisition & Technology)
Director, Test, Systems Engineering and Evaluation
Office of Resources and Ranges
Central Test & Evaluation Investment Program
Pentagon (Room 3D1067)
Washington, DC 20301-3110
September, 1997

Abstract

The purpose of TENA is to affordably and effectively utilize processes and technologies to enable sharing, reuse and interoperability between test and training ranges and resources. This document describes the TENA architecture that will support this purpose.

An architecture is a fundamental and important part of a development approach, which supports reusability of components and the interoperation of diverse systems distributed over multiple platforms and facilities. Development experience has shown the utility of basing architecture upon a Technical Reference Architecture (TRA), in particular in systems where a number of facilities that perform different functions must standardize their implementations to achieve goals such as those that TENA is striving for. The TRA is the mechanism to insure that basic system level requirements critical to interoperability, composability, and reuse are satisfied even when applied to a number of different system architectures. TENA is based on the concept of definition of a TRA that is instantiated as needed to form the core infrastructure for system architectures that serve different T&E and training facilities. The TENA TRA is composed of an Object Model, a TENA Core, and Standards and Protocols. Each of theses are presented in a format most amenable to a clear understanding of their function.

The Object Model represents several views of reality (Test and Training Ranges) that are recognizable and meaningful to range users. The view presented here is primarily a structural model. The TENA Object Model is composed of Classes and the Methods that are available to manipulate or use instances of those classes. In their totality, they comprise the pieces from which a logical range is constructed. The Model is presented in a hierarchical graphical structure accompanied by clarifying text. We have concentrated more on open-air ranges in the initial analysis because of availability of information and cognizant personnel. As we proceed the analysis will extend across the entirety of the domain of T&E and training facilities.

The TENA Core capabilities provide a means for range components to communicate with each other and also mechanisms for the management of the coordinated operation of an instance of the logical range. They are presented in a format that provides a description of each core component (information management infrastructure services and core applications). Certain capabilities that are required for complete TENA operation are presented and defined as Mandatory Applications within the TENA Core. A discussion of the TENA architecture as compared to HLA is also provided. The complete technical details of the services (parameters, exceptions, pre and post conditions, use cases) are contained in Appendix C. Full details of the TENA Core will be elaborated via a series of experiments and prototypes which explore and validate architectural concepts. These details will be provided as an update to this document or released independently as part of the continuing exploratory work.

The Standards and Protocols provide for agreements on issues that support architectural requirements. These involve data representation, communications protocols, supporting platform capabilities, and processes. TENA standards activity has been focused on identification of available standards.

Selection has been largely deferred to development of system architectures and implementations. Experiments and prototypes will allow us to evaluate candidate selections.

Communications standards are a key area of concern for TENA. TENA has supported a modest test of the applicability of ATM technologies, specifically between Edwards Air Force Base and China Lake. A report of this is provided in Volume X. Additional efforts to categorize representations of open air range data, transfer characteristics, and potential standards is also under way at NAWC, China Lake. Results will be incorporated into a revision of this document.

This document is intended for use by system developers, component designers, and other interested parties.

Instructions to the Reader

The Test and Training ENabling Architecture (TENA) fiscal year 1997 Baseline Project Report contains 10 volumes and an Executive Summary. This format provides several advantages. For example, you need not read the detailed technical information in the Technical Reference Architecture (Volume IV) unless you wish. We have provided an Executive Summary which should be read by DoD range management executives and others in a decision-making role. It should also be read as a companion volume to technical volumes. The Management Overview contains enough information from the remaining technical volumes to gain a good understanding of the TENA project background, accomplishments to date, and plans for the future. Additionally, Volume IX, Glossary of Terms and Definitions and Volume X, Other Supporting Information, are intended as companion reference volumes for the reader.

Each volume contains an abstract (all are presented in an appendix to the Executive Summary), Table of Contents, Overview, Introduction, and TENA Project Background. The Overview contains information related to the specific volume, identifies the expected readership, and identifies any relationships with other volumes. The TENA Project Background is the same in each volume. Technical volumes are intended to be "stand-alone" documents that will be upgraded as more information becomes available. An acronym and reference listing, (appendices A and B in every volume) is provided, but for detailed definitions and some cited references, the reader should consult Volumes IX and X.

The TENA Project Baseline Report contains the volumes listed below:

Executive Summary
Volume I - Management Overview
Volume II - Product-Line Approach
Volume III - Requirements
Volume IV - Technical Reference Architecture
Volume V - Logical Range Business Process Model
Volume VI - TENA Application Concepts
Volume VII - Integrated Validation and Verification Plan
Volume VIII - Transition Plan
Volume IX - Glossary of Terms and Definitions
Volume X - Other Supporting Information

The opinions, ideas and recommendations presented in the TENA Baseline Project Report are the views of the TENA Project Team and do not necessarily represent those of the Sponsor.

Table of Contents

TABLE OF FIGURES

1. Overview

1.1 Purpose

This definition of the TENA Technical Reference Architecture describes the foundation upon which the logical range will operate. It defines:

- The component parts (assets) from which the logical range exercise is constructed (Object Model) and the relationships between them,
- The capabilities provided to manage system components, conduct a logical range exercise, and share data (TENA Core capabilities),
- Agreed upon conventions about data representation, communications methods, platform support tools and capabilities, and processes among the facilities and component implementations (Standards and Protocols).

It is applicable to both new facilities and systems designed to be TENA compliant and also to existing legacy systems that interoperate with the TENA Enterprise or will migrate toward TENA compliancy in a planned step-by-step manner. The TENA Enterprise is the set of facilities interoperating via infrastructure services provided by TENA.

1.2 Readership

This document is intended for use by system developers, component designers and other interested parties. It is technical in nature and by necessity presents its information in detail. The three components of the architecture (Object Model, TENA Core capabilities, Standards and Protocols) are presented separately with overviews and introductions. The reader who wishes to understand the architecture being presented, but not become involved in the myriad of details, may read the introduction and overviews of each section and progress to the details only when interested.

1.3 Relationship to other Volumes

This TRA meets the requirements defined in Volume III, TENA Requirements. The Logical Range Business Process Model (LRBPM), Volume V, defines a process to utilize the architectural features to define, plan, schedule, execute and close out an exercise. Volume VI, TENA Application Concepts, describes how the LRBPM and object structures of the TRA work together to support the Logical Range concept. Volume II (Product Line Approach), Volume VII (Integrated Validation and Verification Plan), and Volume VIII (Transition Plan) discuss the support, validation and transition to TENA.

1.4 TENA PROJECT BACKGROUND

1.4.1PROJECT NEED

TENA is part of a coordinated response by the Central Test and Evaluation Investment Program (CTEIP) office to several current and emerging challenges in the test and training range and resource community. These challenges include:

- Reducing software development and maintenance cost,
- Utilizing common instrumentation at multiple facilities,
- Responding to the increased demand for multiple-site exercises and/or exercises which cross T&E/training or live/virtual/constructive boundaries,
- Responding to the increased demand for consistency of information between facilities and across phases of the acquisition process, and
- Capturing critical data to support informed customer and management decisions about resource needs, capabilities, and investments.

1.4.2 PROJECT PURPOSE

The purpose of the TENA project is to respond to these challenges through the establishment of an architecture that efficiently and effectively fosters the sharing, reuse, and interoperability between cooperating Department of Defense (DoD) test ranges and facilities, training ranges, laboratories, and other modeling and simulation activities. The expected synergism will permit efficient and effective testing of new and enhanced weapons systems and will vastly improve the scope and fidelity of worldwide joint/combined training.

1.4.3 PROJECT HISTORY

The Test and Training ENabling Architecture (TENA) project concept was formulated in FY95 by a multi-Service working group. This concept was endorsed by the Test and Evaluation Reliance Investment Board (TERIB), the Board of Operating Directors (BoOD), and the Test and Evaluation Resource Council (TERC).

The Navy is the CTEIP Resource Manager for this project, and has established a Joint Project Office (JPO) for the management of project activities at the Naval Undersea Warfare Center (NUWC) Division, Newport, RI.

Shortly after assembly of the Joint Service Team, several critical observations were made:

- The key to interoperability is not connectivity alone, but rather understanding communications content. This is best promoted by defining an open, object-oriented software architecture that could be used by both legacy and newly built systems.

- The process used to plan, schedule, and otherwise coordinate a multiple-facility, multiple-service exercise must be integral to the development of the architecture, or the capabilities it offers might never be fully utilized.

- The architecture must be conducive to refinement over time and coexists with facility-unique applications. This requires a disciplined architecture development/refinement process. The team adapted the Defense Information Systems Agency (DISA) domain-engineering approach to help develop the architecture and recommends the Product-Line Approach for implementation and life-cycle maintenance.

- Significant investments are being made in other closely related areas such as, Defense Modeling and Simulation Office (DMSO), High Level Architecture (HLA) and the Joint Simulation System (JSIMS) program. TENA must leverage as many of these efforts as practical.

- The TENA concept is radically new to our community. Planning for transition is key to its ultimate acceptance.

1.4.4 STATUS

The project team tested its architecture development process in FY96 producing a "Pilot Architecture." This work was reviewed in several public forums. These reviews were highly supportive of TENA's effort. Two consistent suggestions were that TENA should focus first "on breadth, not depth", and that there should be more emphasis on "problem-space vs. solution-space". These considerations and additional engineering effort has resulted in this refined "Baseline Architecture."

The TENA Baseline contains sufficient detail to continue further analysis and risk reduction efforts and is a good vehicle for discussion, experimentation, and refinement. It is not yet appropriate to use these documents as the blueprint for a major system development. After community feedback, results from risk-reduction prototypes, experiments, and other ongoing efforts are synthesized, the cognizant TENA Baseline documents will be updated as "TENA Rev 0." TENA Rev. 0 will be the appropriate source of design information for a TENA-compliant system implementation.

2. Introduction

2.1 Architecture Development Process

The architecture described in this document establishes a basis for a line of products that integrate training and test and evaluation facilities. It provides levels of structural detail and coordination protocols that allow the facilities to function as an integrated system in a well-coordinated, efficient way and to allow them to schedule and share assets. Enterprise level common characteristics and requirements were extracted from the test and training domain and used in developing the TRA.. The approach TENA will follow is to use the TRA as a means of satisfying enterprise level concerns with instances of this architecture forming the basic structure which is then extended by various system architectures that satisfy lower level concerns.

The TRA is an abstract architecture that provides the structure and coordination capabilities that form the essential foundation around which a complete system architecture will be built. The TRA addresses specifically the requirements that derive from enterprise wide concerns. The TRA is, in effect, an architecture class that is extended by specific system architectures. The inheritance of TRA capabilities by its subclasses (system architectures) is the means by which system wide design decisions are enforced. The current TRA imposes a structure that will be inherited by all future system architectures (such as those for OARs, ISTFs, HITLs or other facilities).

The following diagram provides an illustration of the collection of design decisions, system information, and how that relates to architectural constructs.

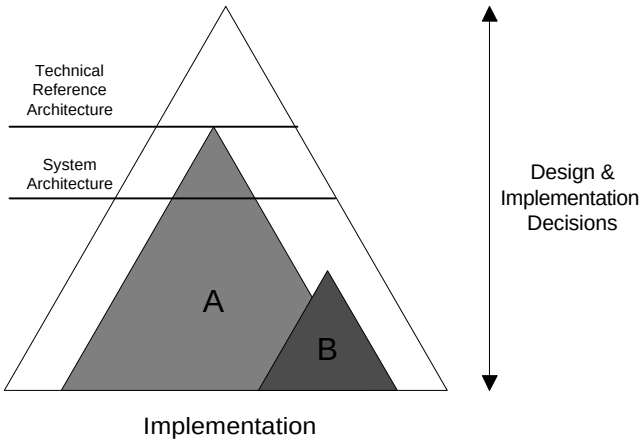

Figure 1. Effects of Design Decisions

The large triangle shown represents the complete collection of design and implementation decisions made about a system within its boundaries. The decisions are sorted so that the most powerful, that is, the decisions that have the greatest effect on the system are located towards the apex and the more local decisions that affect relatively small parts of the system sort towards the base. The region marked A roughly indicates that if a decision was represented as being at the apex of the triangle for A the parts of the system affected lie below it in a similar triangle. The region marked B indicates the same thing but for a decision which is less general and powerful than the decision at A. . The base of the triangle represents the implementation of the system which contains all of the information about the system. An architecture is nothing more than some collection of design decisions about the structure or coordination of a system. Those decisions act as constraints on the further design and implementation of the system. The concern is to make sure that the right set of decisions has been incorporated into the architecture, yet to incorporate a minimal set to prevent designers from being excessively constrained that precludes innovation.

This concept applied to the TENA domain (Figure 2) illustrates where various architectures are situated within this hierarchy and how they relate to each other.

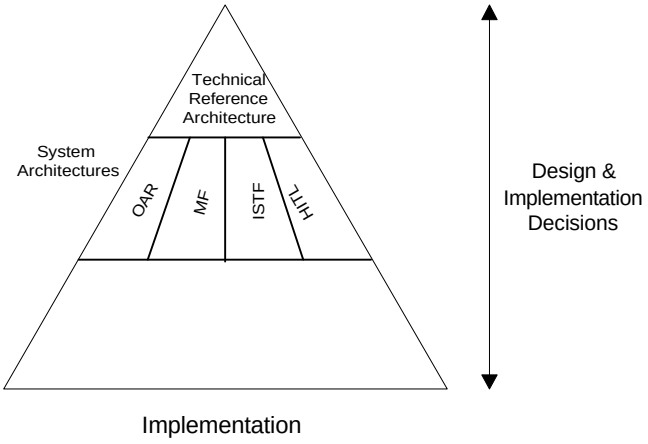

Figure 2. System Architectures

2.2 Architecture Implementation

An architecture can be represented as a horizontal line across the triangle shown above with decisions incorporated into the architecture contained in the region above the line. In actuality this is not a straight line but more of an irregular line with peaks and valleys. These peaks and valleys would represent the current progress that has been made regarding decisions based on current knowledge of the domain and requirements of the system architectures that will follow. For example, some class structures are global and included in the TRA, while others will be developed later as more information is gathered specific to the system architectures of the TENA Enterprise. In Figures 1 and 2, one line is depicted for the TRA and another, further down, for a system architecture. This simply shows that the TRA is itself an architecture, but one which deals with the highest level, highest leverage, most systemically important decisions about the system. The system architecture is realized by adding to the

decisions in the TRA additional decisions that further constrain the implementation to meet important goals and requirements applicable to the specific system being defined. The regions in Figure 2 capture the impact of decisions at various levels in the hierarchy.

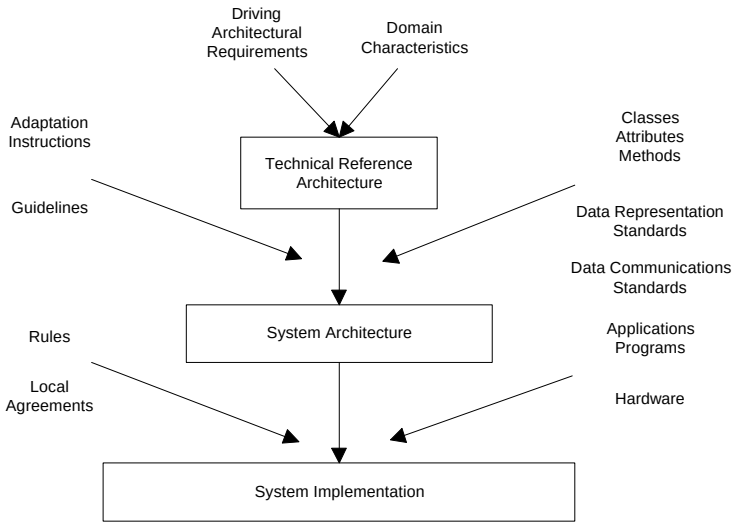

Figure 3 Factors Influencing Design

Locating these lines, that is, determining which decisions to capture as an architecture, involves a series of value judgments by the system architects. The important system issues, external constraints, experience of the architects, etc. all influence where the lines fall. The line between the various levels is shown as an exact, straight boundary. In reality, this line is somewhat fuzzy and irregular reflecting the fact that architects proceed to make decisions to different levels in different parts of the system in response to how strongly system goals are effected by different parts of the system. Certain classes (with objects, attributes and methods), standards and guidelines are included within the TRA, and some will be included in individual system architectures as these systems are analyzed, prototyped and implemented as part of the TENA plan. Figure 3 provides a somewhat different view of the design triangle. It indicates some of the factors which influence design at each of the levels depicted and how they influence the progression from a technical reference architecture to a system architecture and finally to the system's implementation.

The architecture development process is entered with a set of highly derived requirements that respond to the system requirements but are abstracted to a level where only the essential global issues are addressed. Along with these requirements is a set of domain characteristics captured from an analysis or based on the knowledge of expert designers experienced in the domain (Test and Training Facilities). These form the basis for capturing the fundamental patterns of structure and coordination which the TRA expresses. Additional factors are considered when determining characteristics of system architectures and the realization of the implementation. The factors shown are illustrative and not intended to be exhaustive.

Experienced designers know that in developing a large system one should also expect that the design for the architecture which is developed at the start of system design will not survive unchanged throughout the balance of system design and implementation. The validity of the architectural design is proven by subsequent design and especially by building and testing parts of the system.

The Product Line Approach (Volume II) to system development is an integral part of the TENA enterprise. Product lines usually arise when an organization realizes that it has repeatedly produced systems (test and training facilities) that largely serve similar or related customer needs. The reality is frequently that each system, although functionally similar to other systems, is implemented largely from scratch with a unique design. The recognition that components of the various systems can be common among those systems or derived from earlier systems suggests a means to reduce cost and time to market. It also indicates a strategy for reducing variability and, consequently, improving maintainability and reliability. Thus, most organizations evolve into product lines rather then decide to create product lines from scratch.

The product line approach which TENA is following provides a systematic way to identify areas of commonality and where essential system variability lies and exploit that knowledge to both leverage system development (cost and effort) as well as to enforce policies for system qualities such as interoperability. This approach recognizes the need to capitalize on legacy systems and incorporates them or their parts as appropriate within a structure designed to accommodate existing systems.

2.3 Architectural Constituents

TENA is composed of three main constituents. These are the Object Model, the TENA Core, and TENA standards and protocols. These three constituents work together to establish the following:

- A definition of the assets that make up the various facilities including their attributes and methods (as appropriate).
- The manner in which assets and stakeholders associate with each other.
- The identity, structure, and representation of the information which the assets exchange.
- Services and core infrastructure for supporting information exchange and management of operations and the business processes.
- Agreements about the representation of data, operation of communications resources, information display capabilities, and procedures and processes.

The Object Model of TENA describes the structure of and dynamic relationships between components of the resulting system. The Object Model as a whole includes multiple views or models of the system. The discussion in section 3.0 is centered on the view that characterizes the domain or problem space. Other models that will become incorporated over time include the implementation model of the system (solution space), depicting the artifacts created as parts of the system, and the information model of the system, discussed shortly.

The top level view of the architecture from the implementation perspective is shown in Figure 4. The TENA Enterprise is a class that represents the collection of facilities that are participating in a

common set of activities related to the intended uses of the facilities as a cooperative set of interacting assets.

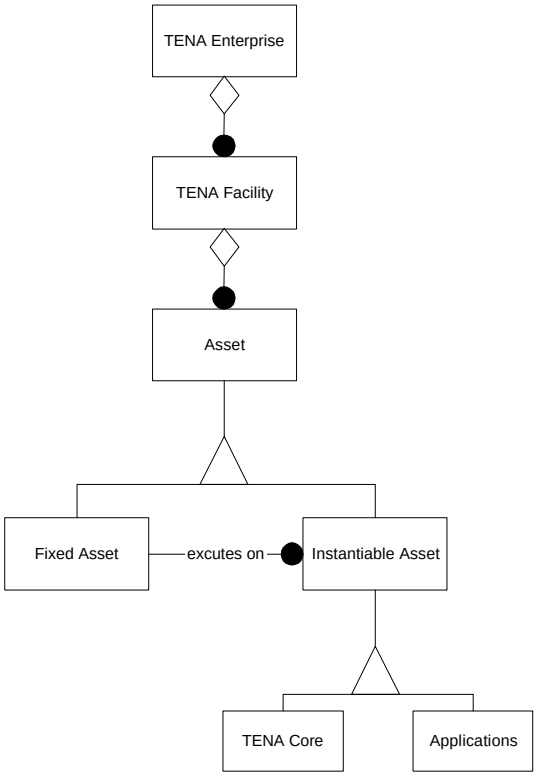

Figure 4. Implementation Perspective View

The facility class corresponds roughly to our current notion of a range, laboratory, or other T&E or training facility. In the domain view this corresponds to the subclasses of the Logical Range Resources class and Mission Space class. This class is an aggregation of assets that comprise the facility. Assets include all material, geographic space, equipment, personnel, etc. which facilities use in their day-to-day operations. It may include equipment they own and organic personnel as well as equipment or personnel they contract for.

Assets come in one of two types. There are fixed assets have a persistence and a physical existence. Instantiable assets are those which are transient. Instantiable assets include software components that execute on various fixed assets or communications assets which are allocated from some pool of resources for a temporary period such as communications circuits provided by common carriers.

The domain model which is documented in the following Object Model section is not orthogonal to the implementation model. Since the domain model describes the assets which are used by facilities there are sections of the domain model which are coincident with the implementation model. Figure 5 shows a portion of the domain model that elaborates the structure of the Logical Range Resources class. The subclasses of this class, which include the Open Air T&E/Training Range class, aggregate various

resources. These resources, from the domain perspective, are assets from the implementation perspective.

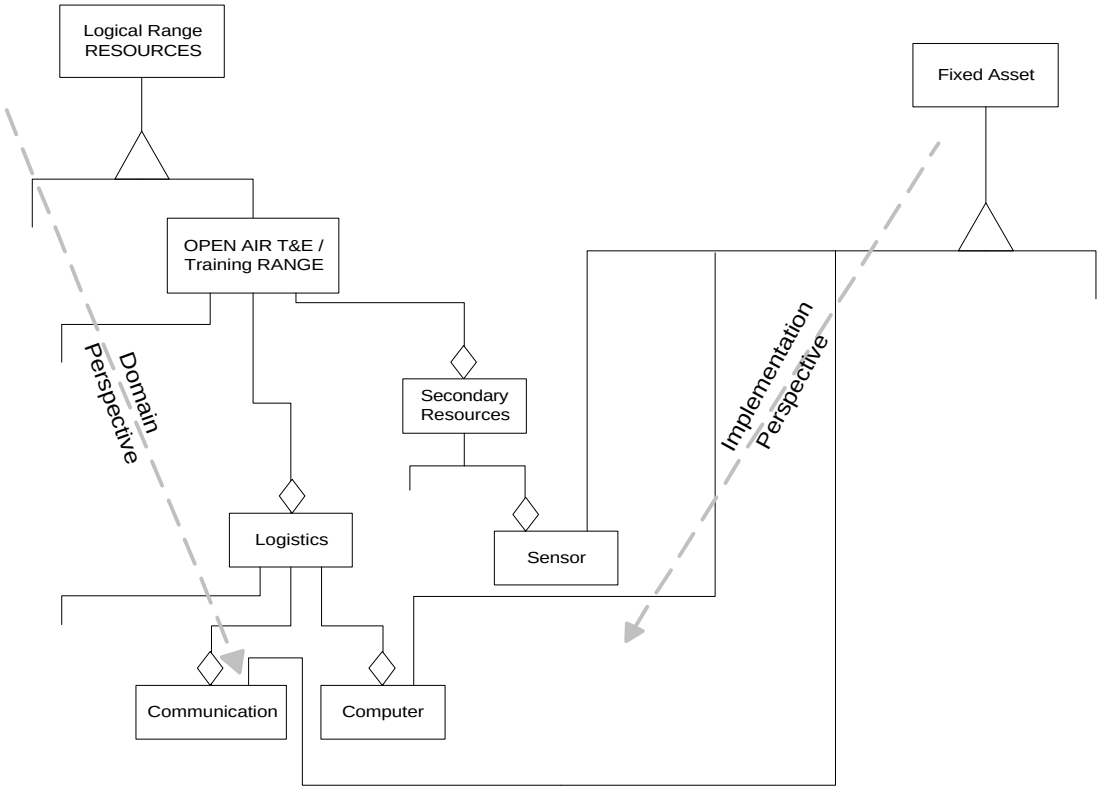

Figure 5. Domain Model

The TENA Core, described in some detail below, constitutes the infrastructure for the TENA Enterprise.

Various standards will be specified at different levels within the hierarchy of the architecture during the process of developing TENA and the systems that it supports. Many of these standards will be specified at the level of the system architecture and where system architectures are defined to support specific types of facilities. Other standards will be specified at the implementation level. These standards are specified in response to the need to support certain required technologies, platform provided capabilities, information processing conventions, and information exchange resources. Since these technologies and resources evolve over time and are replaced as they become obsolete, the standards which are incorporated into TENA will evolve in response to the changing needs of facilities. A conscious effort has been made in defining the architecture to isolate places where applicable standards are subject to evolving at a relatively rapid pace and encapsulate those locations with well-defined and stable interfaces. This results in an architecture where standards that have wide applicability across the architecture (at the level of the TRA) are restricted to those known to be stable over time. In general, these broad standards are the direct result of TENA design decisions and are promulgated by the various design teams.

Volatile standards are more likely to be required for support of technologies and components encapsulated at lower levels of the architecture. For example, communications standards which specify the data representations, signaling protocols, circuit management protocols, etc., apply to system architectures by defining aspects of the implementation of certain infrastructure components that directly interface to the communications resources provided by the facility. This strategy defers most of the standards specification to the design of system architectures, component design, and component implementation. Readers will not see many standards referenced at this point because of this strategic approach. As development proceeds, additional standards will be incorporated.

2.4 Principles of the Architecture

2.4.1 Constrained Composition

One of the goals for the system resonates with a theme that is universal to the infrastructure. This is the idea of a highly flexible ability to compose the system or segments of the system for specific intended purposes that may be either transitory or permanent in nature. The notion of configuration of a logical range, which is a collection of facility assets assembled to perform specific functions dictated by user requirements, is basic to this theme. Assets which provide capabilities required to generate the products which facility users need are selected from the set of facility assets available. Schedules are negotiated to obtain access to the assets to match user time windows and the logical range is instantiated to realize the planned exercise (execute the test plan defined). A number of constraints apply to the use of assets including: physical proximity and location, coverage regions, performance capabilities, and subsystem compatibility. These constraints restrain system users to meaningful compositions capable of producing useful results.

2.4.2 Dynamic, Run-time Characterization

The logical range is intended to respond to many allowable compositions and permit the fairly rapid reconfiguration of itself during operation. Because of this fact and the wide variety of asset types available, the multitude of possible coordination paths, and the numerous and changing data representations managed it is very difficult to imagine a system architecture responding to these conditions without the ability to self describe assets and data.

Components of the infrastructure will be required to deal with assets on the basis of descriptions of their capabilities and characteristics provided by them. The architecture will also establish methods to allow the self description of data representations which can be provided either prior to or concurrent with data transfer. Under many conditions components will negotiate representation issues before operation commences thereby avoiding execution penalties during test operations. As required, standards are established for parameters of services.

This theme should be followed through applications such as allowing for dynamic development of displays where required or desired. Much of this is specified by the architecture within specific product development frameworks.

2.4.3 Subscription Service

Use is made of an object based approach for subscription services for data access. Producers of data announce to the infrastructure their intent to publish certain data and they describe the characteristics of that data. Users of data make known their need by announcing subscriptions to specific data. The infrastructure instances negotiate among themselves to enable delivery of the required data in a timely manner. The parties to the data transfer can prescribe quality of service parameters which will be used by the infrastructure to configure and assign communications resources. Parties may request changes to quality of service during operation to accommodate changing needs. Actual data transfer is accomplished according to protocols established for use of quality of service levels.

A primary intent for introducing this manner of establishing interfaces is the need to make efficient use of resources. This is true in a temporal sense by allowing for a efficient matching of latency requirements to the characteristics of available transfer mechanisms. It is also true in the sense of managing capacity restrictions. Data should be communicated only if it is required, at the rate it is required, for the period it is required, and, in most cases, only when it has changed. However, provision is also made for certain continuous data streams which do not fit this model. The architecture supports this though a separation of control operations from data transfer operations. Filtering of data on the basis of interest sets, regions of interest, and rate of consumption, on a user specified basis, supports managing resource utilization. A wide variety of implementation options can be provided for.

2.4.4 Controlled Information Access

Most information within the enterprise is shared via use of Distribution Services. Any data passed between applications travels by this route regardless of the identity or location of the application. For a number of reasons, including security and the need to protect information that is internal to a specific facility, the architecture provides controlled access to information that is published. Levels of access allow users to limit information access to a desired subset of all users. Information that is published with a global access tag is available to any user at any location. Infrastructure services will enforce an access control policy based on instructions provided by information publishers.

The current assumption is that, initially, systems will operate in a *system high* configuration with respect to information security. However, the potential impact on the system and its architecture to support a true multi-level secure (MLS) system is significant enough that the need to support this more involved security configuration will be considered in the architecture. It is unlikely that we would attempt to build components expressly intended for supporting MLS or providing trusted components in the infrastructure until necessary but we see a need to make sure they are planned into the architecture so they can be smoothly integrated when required. Many of the impacts can be relegated to implementation time decisions if the architecture provides hooks and has anticipated their influence.

2.4.5 Negotiated Quality of Service

It is apparent that some services that the infrastructure makes available have significant performance and cost implications. In particular, communications include data streams with especially large capacity requirements or strict latency tolerance. In addition, reliability is a factor in many cases. The

architecture recognizes the need to match specific physical assets to special performance figures of merit and includes protocols for negotiation about these assets. Users of services, where appropriate, can request that specialized assets be allocated when needed for operational reasons.

These protocols rely on the principal of separation of control information from data. This permits the infrastructure to establish well-known and characterized control paths to establish data paths. Standard interfaces are provided for applications with the infrastructure managing connections between those interfaces and the underlying physical components. Applications are usually ignorant of the specific details of transfer of the data. This, however, does not mean that there will not be specific and unique processing required by applications when using special dedicated circuits. This is a way of negotiating with the infrastructure about how much functionality will be provided by infrastructure components. If the infrastructure is not tasked to provide required functionality the application will have to provide it.

Infrastructure services make applications simpler, more maintainable, and more portable when they provide uniform functionality to applications. However, the architecture recognizes and supports the need for matching capabilities to performance constraints.

2.5 High Level System Architecture View

2.5.1 The TENA Enterprise

The TENA Enterprise is the collection of TENA compliant systems that are joined together via appropriate communications facilities as an integrated, interoperating system of systems. As facilities bring their TENA-compliant systems on line they determine whether the systems are to join as part of the TENA Enterprise or to operate in an isolated, stand-alone mode. The architecture specifies a protocol used by these systems to announce their intent to join the enterprise system. Other systems which are joining or have joined the enterprise system respond to an announcement of intent to join (or resign, as appropriate), establishing a distributed and fully replicated enterprise state.

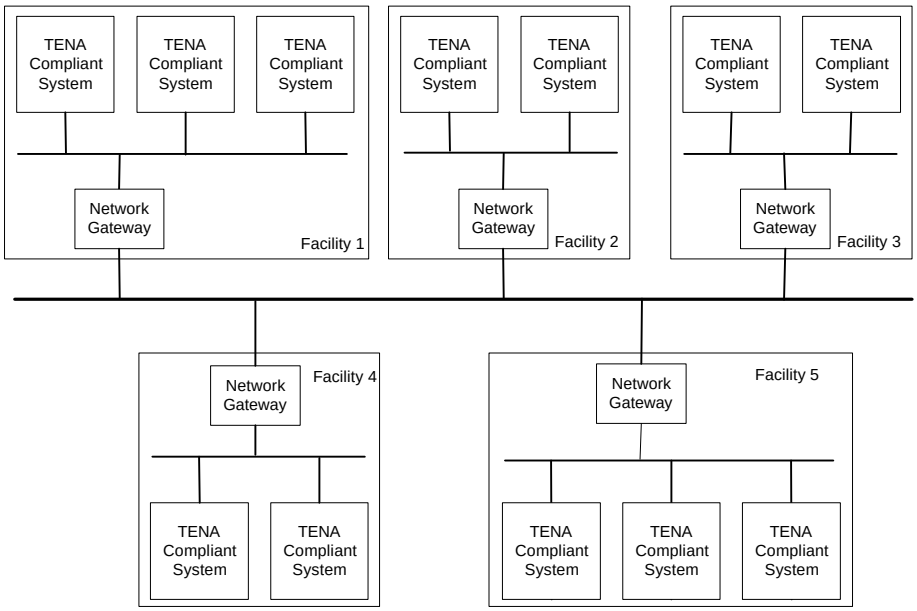

Figure 6. The TENA Enterprise

All facilities and all systems within the enterprise are able to recognize and interact in a meaningful manner with all other systems that have joined the enterprise. The network connections required to join the enterprise are predefined and well known to all TENA compliant systems. Individual systems can join or resign as required and no single system or set of systems are responsible for control of or maintaining the state of the system as a whole. Maintenance of the state of the enterprise is fully decentralized and distributed.

Compliant systems will join the enterprise when they wish to make themselves and their assets available for use by the enterprise as a whole or if they wish to make use of assets which are located at other facilities. While Figure 6 indicates the existence of a single TENA enterprise, there is no inherent limitation on the number of separate and concurrently operating enterprises. If conditions dictate that multiple enterprises be constituted for different purposes the protocol for establishing the enterprise supports this. Facility managers will have to provide information on which communications facilities to use.

There are two effective levels of compliance for assets or resources owned by facilities and the facilities themselves. Assets that are based on computational equipment and that can interoperate with other such systems are rated on their individual compliance. Facilities are rated on their compliance based on the aggregate compliance of systems described above and by whether they make available other assets (not based on computational systems) available for enterprise wide use. For example, a facility may consist of, among other assets, terrain over which test or training exercises may be conducted. That terrain asset can be made available to the enterprise by its being described within the master asset catalog and providing for the ability of users to schedule or reserve its use for an exercise. Thus if the assets of a facility are made accessible to the enterprise the facility itself is TENA-compliant at some level. There are different levels of compliance described in Volume VIII Transition Plan, and provision

is made for many different ways to expose assets for enterprise use based on the desires of the facility owners or managers.

2.5.2 The Facility

Facilities are collections of assets that are located in relatively close proximity and under the control of a single command structure. They correspond to the individual facilities, and laboratories that operate today. The facility contains a number of assets some of that are based on computational platforms and some which are not. Assets based on computational resources that are capable of individual compliance have a structure similar to that shown in Figure 7.

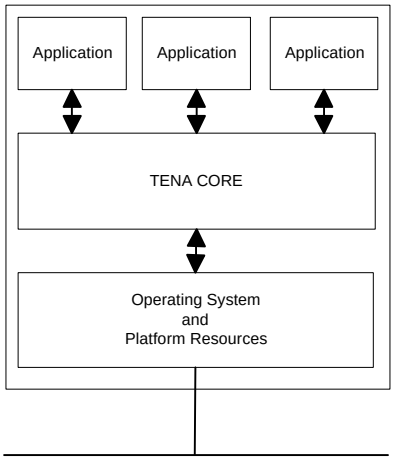

Figure 7. A TENA Compliant System

The TENA Core is the building block of the system infrastructure. Each system has an instance of the TENA Core that encapsulates the underlying platform. It provides isolation for the details of what resources are available and their specific operating characteristics, as well as platform peculiar representations, protocols, etc. It also provides those basic services and core applications which enable the system to operate as a part of the enterprise. It is responsible for managing information about which assets are available and their characteristics, as well as providing for the instantiation and control of the logical range. The logical range is a temporary association of a set of assets for the purposes of conducting a specific test/exercise. The TENA Core also provides additional services that provide for exchange of information, configuration of test/exercise assets, set up and operation of communications resources, and other services required to manage the enterprise and its facilities. The TENA Core is described in detail below.

The TENA Core maintains information about assets that are available to the enterprise. This information is stored in a database called the Master Asset Catalog and is available to any application by using services exported by the TENA Core. This information is used in managing the enterprise and in defining test/exercise plans. The Core also maintains a database of information classes that form the

basis for sharing information within the enterprise. This is maintained in the Information Class Catalog that is available to any application through services exported by the TENA Core.

Each facility consists of one or more assets. In general, there will be at least one asset that is based on a computational platform and is capable of joining the TENA Enterprise and responding to information distributed within the enterprise. At least one of the assets in a facility is designated a caretaker. This means that the asset is responsible for representing the assets for which it is responsible to the enterprise and performing some of the management associated with the assets such as scheduling their use. The number of caretakers and the allocation of assets to caretakers is determined by facility managers in response to their local requirements.

3.0 TENA Object Model

C:\tmp\3.doc

This section defines the Baseline TENA Object Model. It is a conceptual view of the architecture components (classes), their operations, attributes (data items) and relationships. It is assumed the reader has some familiarity with object-oriented analysis techniques [Blaha, 1991].

The structure and components of this object model were selected to respond to primary TENA needs (listed below) and solutions (sharing, interoperability, and reuse):

3.1.1 Reducing software development and maintenance cost. (Reuse)

The object-oriented approach (together with the Product Line Approach - Volume II) allows us to reduce software development and maintenance cost. All of the externally visible information about a major component is collected in a package which includes the operations (methods), data (attributes), and relationship to other components. This promotes software reuse by having clearly specified interfaces while still allowing for implementation flexibility. The architecture does not force, but strongly supports object-oriented implementations. Object-oriented implementations promote software maintenance because they isolate the impacts of a single software change. The impacts are clearly specified in the object definition.

3.1.2 Utilizing common instrumentation at multiple facilities. (Reuse, Interoperability, and Sharing)

Details of hardware interfaces are hidden and encapsulated to isolate the impact of changes. Software interfaces are standard objects which can be reused. The impact of changes to instrumentation, both hardware and software is controlled and predictable using a combination of object-oriented techniques and specification of standards at the appropriate architectural level.

3.1.3 Responding to the increased demand for multiple-site exercises and/or exercises which cross T&E/Training or live/virtual/constructive boundaries. (Interoperability and Sharing)

The scope of the architecture is from customer entry to customer exit. This scope was chosen because it is the highest level abstraction that is common to all participating test and training ranges and resources.

Decisions about which environments and platforms to use in conducting an exercise are made early in the life-cycle of the exercise. In order to "seamlessly interoperate" across multiple sites/facilities and/or across live/virtual/constructive boundaries, the architecture must have visibility into where those decisions are made. These decisions take place early in the planning of an exercise. A common understanding of the process (the Logical Range Business Process Model - Volume V) and the components (this object model) which support an exercise is required for routine construction of mission spaces across physical and temporal boundaries. A common view of the mission space is required to execute these exercises.

3.1.4 Responding to the increased demand for consistency of information between facilities and across phases of the acquisition process. (Sharing)

The object model defines data elements for consistency of information. There are two approaches to information consistency, both supported by the object structure. The first approach is to agree on a community-wide definition. The second approach is to agree to a mechanism of providing the definition. Objects and their data element representations can be self-defining, i.e. support tools can determine the number, name, and representation of data elements by agreeing on a common object description language. For example, an information presentation tool could query the "ship" object to determine how many parameters are available for display, the frequency of their update, and format of the data. It could also determine the valid operations which could be performed on this particular ship. The tool could adapt to multiple ship objects without needing to be reprogrammed for each new type of ship. The same ship object could be used across multiple phases of the acquisition process to promote consistency of information, even if the not all data or operations are used by every phase. TENA plans to recommend an object description language in FY98.

3.1.5 Capturing critical data to support informed customer and management decisions about resource needs, capabilities, and investments. (Sharing)

Collecting critical information for planning investments and tracking resource utilization is time-consuming, expensive, and sporadic. When data is collected it is difficult to compare from one facility to another. Architectural support for collection of this data requires visibility into the customer requirements, documentation of which requirements are being met or not met, and utilization information for a wide variety of assets. The object model provides structures (object classes) where this data can be saved, collected, and analyzed together with all supporting documentation. Policies for how much of this data is collected and how it is used are under control of individual facility operators.

The TENA Object Model will not look like a typical OAR, HITL, or ISTF. Classical architectures in our community can be described as in CTTRA:

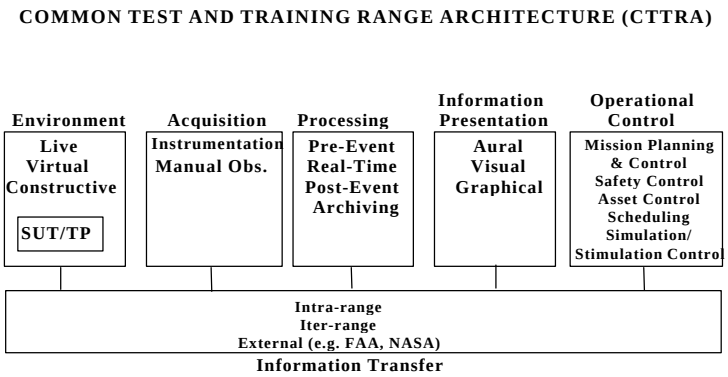

Figure 8. CTTRA Functional Architecture

This is indicative of an abstraction of how our systems work, i.e. the solution-space of our problem domain. In fact, it is indicative of only a small piece of our solution-space, the piece focused on real-time data collection, processing, display and subsequent post-test analysis.

TENA has focused on our problem-space, i.e. what our customers do with our systems. The scope of the architecture is from customer entry to customer exit. This is no small change. It takes our traditional engineering focus and makes it a supporting cast member in the fundamental problem we all share: Defining, Scheduling, Planning, Executing, and Closing test and/or training exercises which meet specific customer requirements and objectives. The object model provides structures which support these functions.

These structures can be used to capture and reuse customer requirements, scenarios, plans, and configurations, and to provide a context for comparison of results among different executions of an exercise. An introduction to the primary structures which capture this information and the relationship between them is offered below.

Mission Space [The term mission space is used in preference to battlespace to include operations other than war and other non-battle missions.]

The mission space describes the environment, platforms, and events required to support an exercise. Examples include open-ocean, desert terrain, submarine, and tank. Components of the mission space can be provided by multiple potential sources.

3.1.6 Instance of the Mission Space

A specific environment, platform, or event (whether live, virtual, or constructive). We also refer to this as a Primary Resource. An example is that the Pt. Mugu Sea Range may supply the Open Ocean Environment required in the Mission Space.

3.1.7 Secondary Resources

A collection of resources which provide the information or actions required by the Primary Resources. For example, the position of a ship may be a required information element from the mission space. The Pt. Mugu Sea Range can provide positional information from many instrumentation sources. Sometimes the specific source is critical to the exercise, at other times the facility is free to choose which instruments are best suited to the task based on local considerations. These kinds of decisions are "secondary" with respect to the primary goal of determining a participant's position, although they are certainly not "secondary" in importance from the point of view of range operations.

3.1.8 Logistic Resources

The equipment and personnel that support both Primary and Secondary Resources and other support activities are Logistic Resources. All TENA-compliant facilities are assumed to have some basic computational power, communications capabilities, financial requirements, and personnel to support the exercise. The assignment of these resources to support a particular test is dependent on both the primary and secondary resource choices. Multiple sets of logistic resources could handle the same primary and secondary resource requirements. Logistic resources may be used to provide additional support for an exercise being conducted largely at another facility.

3.2 The TENA Object Model Diagram

This section describes the TENA Object Model, in particular the structure of the object classes. Portions of the model are reproduced in the text as they are described. The complete model diagrams can be found in Appendix E. These descriptions will be elaborated and modified when necessary as the TENA Object Model matures.

The TENA Object Model is one portion of the Technical Reference Architecture(TRA). The TRA is used to develop domain-specific architectures for each community served by TENA (OAR, HITL, ISTF, MF, etc.). TENA Applications Concepts (Volume VI) describes how this structure is used together with the Logical Range Business Process Model (Volume V) to support the concept of the Logical Range. Section 3.2 offers an expansion of the TRA level model for the domain of Open Air Ranges. Within each domain, the model may be further customized for specific installations. Section 3.3 explains the Information Presenter class at a detailed level. A primary requirement of TENA is to offer a flexible presentation/user interface which can dynamically build a wide-variety of displays and/or user queries for any TENA-compliant system. This capability is fundamental to TENA. We provide more detail in this area to support early review, discussion, and prototyping of this capability. Section 3.4 provides an example of how the Information Presenter class would be used to build typical OAR displays/user interfaces.

In the diagrams that follow, the Logical Range Support Tool is highlighted to indicate that it is not an object class, but a set of applications.

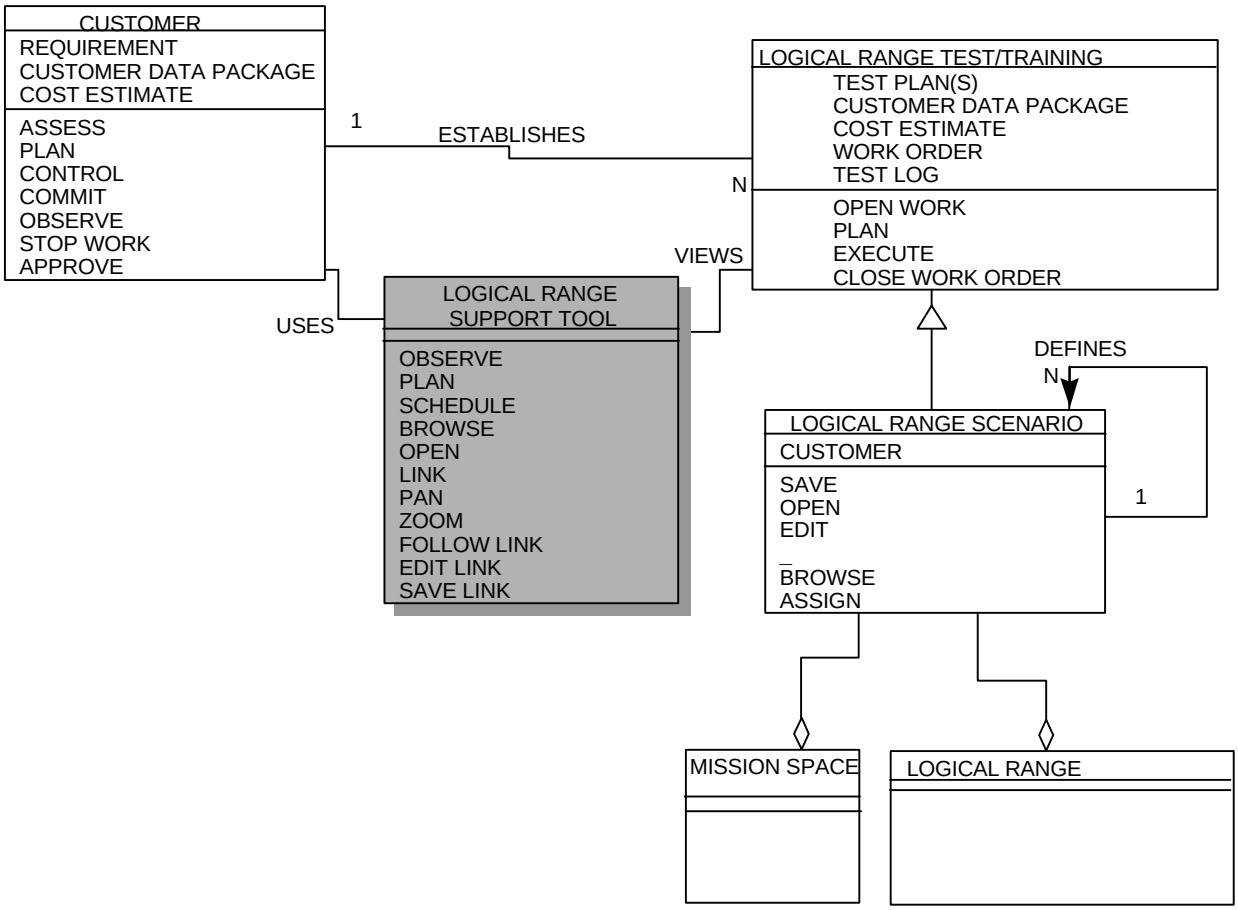

Figure 9. TENA OM-Level Zero

3.2.1 Customer

The customer class represents the person, command, or organization that reserves or sponsors specific test and evaluation and/or training activities at the logical range.

A customer determines that the requirement exists to conduct a test or training event, what type(s) of test is desired and/or what type/degree of training is desired and establishes criteria for the planning, scheduling and evaluation of the desired event. The customer can assess, observe and control the exercise or test. The customer receives a cost estimate for the exercise or test and commits the exercise or test based on that estimate and the available budget.

The customer receives a customer data package at the end of the exercise or test reflecting the data gathered and evaluated at the test points covered during the conduct of the exercise or test.

3.2.2 Logical Range Support Tool

The Logical Range Support Tool is an application set, defined as mandatory for the TENA Architecture, which provides the mechanisms required to define, plan, schedule and execute a Logical Range Test/Training Exercise.

The Logical Range Support Tool provides the ability to browse the Logical Range Resource Catalog, which contains definitions of all assets which are visible to TENA.. These assets include Test Plans and Logical Range Scenarios, as well as the resources required to conduct a Logical Range Test/Training Exercise.

Logical Range Support Tool is highlighted in the object model diagram to indicate that it is not an object class but rather a set of applications.

3.2.3 Logical Range Test/Training Exercise

The Logical Range Test/Training Exercise is a dynamic entity, composed of requirements, plans, schedules, and resources required to test a system or system component, or to train personnel as required.

The Logical Range Test/Training Exercise is driven by a set of test plans and schedules, and as a result of the conduct of the Logical Range Test/Training Exercise, a customer data package is produced reflecting the data gathered and evaluated at the test points covered during the conduct of the exercise or test. This customer data package is delivered to the responsible customer for evaluation and review as to the effectiveness of the Logical Range Test/Training Exercise.

3.2.4 Logical Range Scenario

A Logical Range Scenario is a specific implementation of a Test Plan to meet the Customers requirements.

Many different Logical Range Scenarios are possible to meet the Logical Range Test/Training Exercise requirements. Some or all features of the Logical Range Test/Training Exercise may be simulated, or conducted with "stand-ins" when assets are not available or are too expensive to expend.

3.2.5 Mission Space

The Mission Space comprises those primary resources that are the focus of the Logical Range Scenario.

These primary resources include the participant, or item under test, the environment, either real or simulated, and the specific events that will define the Logical Range Test/Training Exercise.

3.2.6 Logical Range Resources

The Logical Range Resources comprise those Secondary and Logistics Resources that are required to implement a specific Logical Range Scenario.

Example Secondary Resources for an Open Air Range include sensors, stimulators, analyzers. Logistics Resources support the Secondary Resources and include computers, communications links, personnel and financial support.

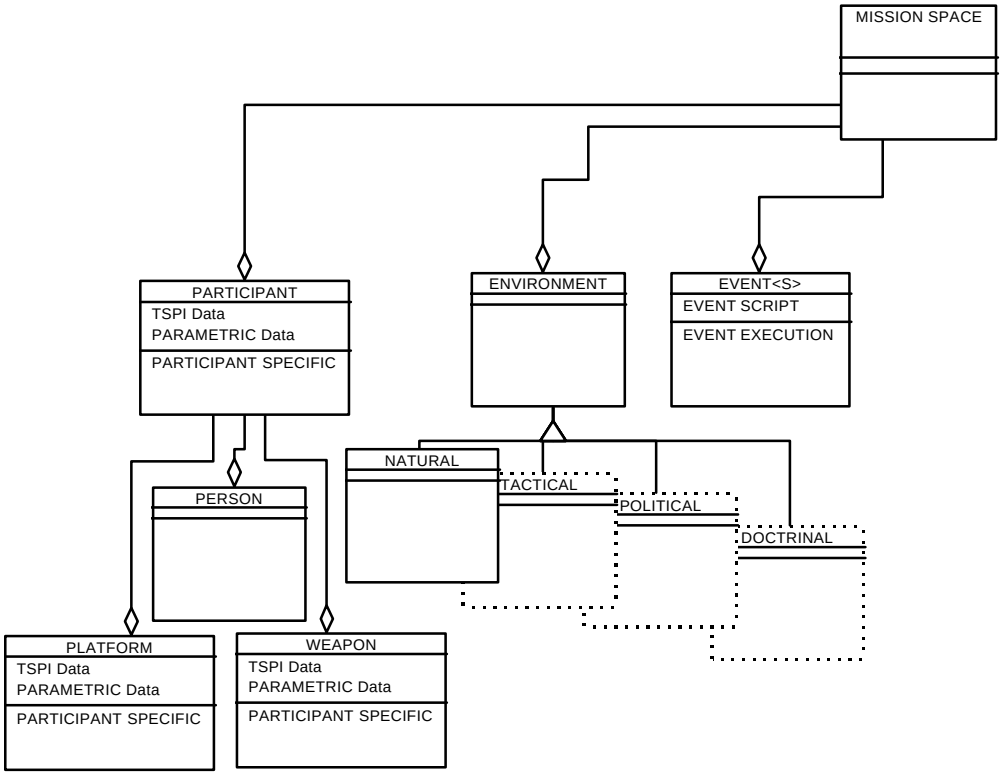

Figure 10. Mission Space Class

3.2.7 Participant

The Participant is the focus of a particular Logical Range Test/Training Exercise. Instances of the participant class are those items that the range and/or facility is monitoring.

A participant is an abstraction of those properties of a platform or any other item/participant of direct concern to the exercise.

It is not the intent of TENA to model every possible participant, but rather to establish a structure which will be understood by any TENA compliant system. Details would be supplied by customers or be available in DOD libraries.

3.2.8 Environment

The conditions under which the Participant is immersed during the execution of a Logical Range Test/Training Exercise.

The implementation of the environment may eventually be assigned to real, simulated, constructive, or mixed sources. Different aspects of the environment include natural, tactical, political, and doctrinal components.

3.2.8.1 Natural

The Natural Environment Object Class refers to the physical environmental characteristics required to support an exercise. These include: Terrain, Electromagnetic Environment, Ocean, Atmosphere, and Space.

3.2.8.2 Tactical

The tactical environment is a characterization of an engagement situation between forces based on historical and/or analytical experience in similar situations. Example: Amphibious assault on a heavily guarded beach head with a large non-combatant population nearby.

3.2.8.3 Political

The political environment is a characterization of the relationship between actual or potential participant groups in a scenario. Examples could be as simple as noting that two of the three participating forces are at peace and committed to a mutual defense treaty, or that in an operation other than war, that the primary goals of the hostile takeover of a military building are to embarrass the President and the "protesters" have never been known to harm an occupied building or building occupants.

3.2.8.4 Doctrinal

The doctrinal environment is a characterization of the "rules of engagement". Forces operating under different rules may produce different results even if all other environmental factors are constant.

3.2.9 Event

An event is an expression of scenario conditions in the environment and/or of the participants which are of special interest to exercising the scenario. Some example events include: Aircraft within 200 NM of surface ship, Tank ready to climb incline. Tank stopped on incline. Ambient air temperature is 98 degrees. Signal strength on channel 21 is greater than or equal to 9 dB. Weapon away. Events can be compounded conditions: Target acquired and no threats within 200 NM.

In the following diagram the facilities (Open Air Range, Simulation, Integration Laboratory, ISTF, HTL, and MF) are outlined to indicate that they are currently only a notional representation in the object model. Efforts to date on the TENA Object Model have concentrated on analyzing the Open Air Range. Subject Matter Experts are confident that the completion of the required analysis of other facilities will verify that, for the purposes of TENA, they have a similar object class structure. Once the supposition that there is a similar class structure is verified, facility specific details will be encapsulated by other object classes.

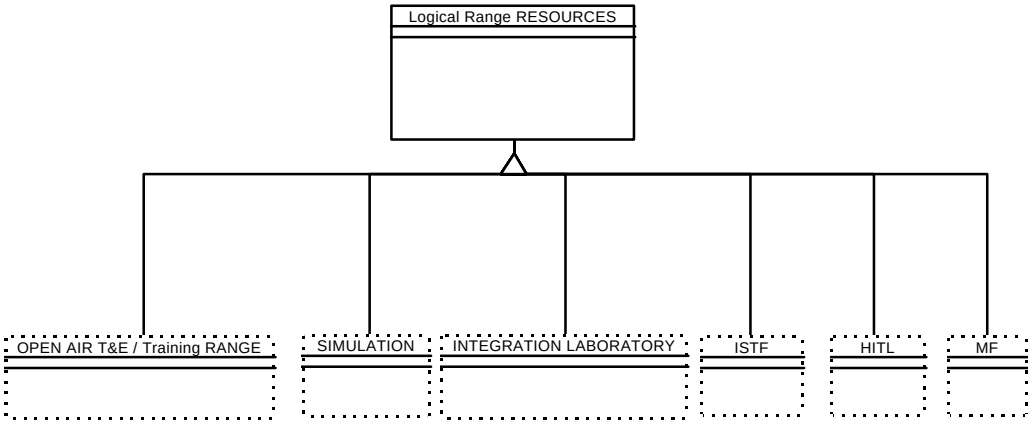

Figure 11. Logical Range Resources Class

3.2.10 Logical Range Resources

The Logical Range Resources are a collection of individually schedulable assets from one or more participating facilities, used to support the execution of a Logical Range Test/Training Exercise .

Participating facilities may include T&E/Training open-air ranges and the logical range visible assets of other peer T&E/Training domains including: Simulations, Integration Laboratories, Installed Systems Test Facilities (ISTF), Hardware in the Loop (HITL) Facilities, Measurement Facilities, and Tactical Units. This combination of assets meets identified customer needs and requirements for a specific Logical Range Test/Training Exercise.

3.2.10.1 Open Air T&E/Training Range

The Open Air T&E/Training Range is a combination of the physical open-air range assets available to participate in the Logical Range Test/Training Exercise.

Open Air Ranges consist of controlled or restricted areas to support the test of platforms/systems in a real world, dynamic environment. They are instrumented with data collection, time-space-position information, positive control of test participants, and real or simulated targets and threats as appropriate. They, unlike the Logical Range, also have a permanent management structure and individual identity that exists regardless of the existence or conduct of any Logical Range Test/Training Exercise.

> Field or Open-air Test Events refers to any test conducted in an open environment. It includes surface (land and sea), undersea, airborne and spaceborne testing. Field tests are conducted where it is feasible, safe and secure to test all or part of the SUT[participant] in an environment that is normally more realistic than any attainable indoors. Field tests may allow the SUT[participant] to be operated more closely to its operational conditions. However, particularly with EW systems, field tests may provide less insight into the performance of a system because indoor facilities are the only place high density, high fidelity threat signals can be generated in a secure environment.[1]

3.2.10.2 Simulation (Digital Models and Computer Simulations - DMS)

Simulation refers to those assets of models and simulations which are "visible" to the logical range.

These models and simulations either provide a simulated test environment or representations of systems, components, and platforms. DMS's are used throughout the development, test, and training process as analytical tools, as well as tools to drive or control electronic and other environmental stimuli.

DMS's can be utilized to create or modify the event environment in order to assist in the overall development of a realistic environment into which participant is immersed. This may involve the creation of physical geographic environments and environmental modifiers, the insertion of simulated forces and force actions within a real or created environment, or a combination of the two.

DMS's "can be used to design a better test program, add realism to test scenarios, extrapolate results of testing, and explain aspects of system performance observed during testing. This can reduce time, resources, and risk to an acquisition program. In some cases, M&S is the only way to conduct system assessment and is the only way to generate "reproducible" scenarios and conditions."[2]

> Simulation/Stimulation Events are used extensively in the DoD test process. They can be applied to computer or physical working models or the SUT [participants]. They may be real time or non-real time models. Effective use of credible models and their simulation/stimulation events will provide cost effective T&E.[3]

3.2.10.3 Integration Laboratories

These are the integration laboratory assets which are "visible" to the logical range.

An Integration Laboratory is a "facility that supports the integration of system components and/or software in a laboratory environment for development, experiments, and testing. The integration laboratory "simulates" (or replicates) a system to a known extent and allows the modification/addition of component hardware/software for use without many of the restrictions or difficulties that would be encountered using actual system hardware or host platforms."[4] Integration Laboratories are generally

---

1 The DoD Test and Evaluation Process, June 13, 1994, 2.2.2.2.6

[2] Simulation, Test, and Evaluation Process (STEP) Guidelines, December 2, 1996, Draft

[3] The DoD Test and Evaluation Process, June 13, 1994, 2.2.2.2.7

[4] Simulation, Test, and Evaluation Process (STEP) Guidelines, December 2, 1996, Draft

platform specific or unique. However, the simulated stimuli and data collection capabilities required by Integration Laboratories are often common with those required by HITLs and ISTFs.

> Integration Test Events test components, subsystems and systems combined with other elements. The other elements may be other parts of the same system or other systems with which the SUT [participant] must operate. These tests are frequently conducted in integration laboratories specifically designed to test the SUT [participant] integrated with other systems or functions. Integration laboratories are generally weapon system specific and are used from the beginning of a system's development through integration and fielding. These tests employ a variety of models, simulations, and stimulation's to generate scenarios and background at or near real time.[5]

3.2.10.4 Installed Systems Test Facilities (ISTF)

These are the installed systems test facility assets which are "visible" to the logical range.

> "ISTF are facilities where entire systems or sub-systems get their first workout in the environment in which they will operate (e.g., inside an aircraft). A full capability ISTF has the ability to mix a complete spectrum of players from synthetic (digital models) to real (actual hardware) to hybrid (a combination of both); the ability to provide multilevel threat simulations (open-loop and closed-loop signal simulators, including actual or simulated threat system hardware); and the ability to provide simulations of all C3 elements a system would be expected to operate in the real world. The Navy's Air Combat Environment Test and Evaluation Facility (ACETEF) at Patuxent River, MD is an example of an ISTF."[6]

> Installed Systems Events provide capabilities to evaluate SUTs [participant] that are installed on and integrated with their host platforms. These tests can occur in indoor facilities such as electronic warfare (EW) or climatic chambers or as outdoor DT and OT tests. Chambers provide a secure site to evaluate the capabilities and limitations of the system against simulated and stimulated inputs. Climatic chambers examine SUT[participant] capabilities in varied temperature and humidity conditions without having to transport the SUT[participant] to those naturally occurring climates.[7]

3.2.10.5 Hardware in the Loop (HITL)

These are the Hardware in the Loop facility assets which are "visible" to the logical range.

These facilities provide capabilities to test systems or their components at various stages of development (e.g. brassboard, breadboard, prototype, pre-production, production). HITLs provide stimuli and data collection capabilities to permit test and evaluation of a system/component independent of the host platform.

> Hardware-in-the-loop (HITL) Events use elements of the SUT [participant] in combination with software to examine the performance of those elements before the entire system is available or when a specific capability cannot be tested. HITL events, such as breadboard, brassboard, or prototype tests permit system/subsystem evaluation during various stages of development.[8]

---

[5] The DoD Test and Evaluation Process, June 13, 1994, 2.2.2.2.3

[6] Simulation, Test, and Evaluation Process (STEP) Guidelines, December 2, 1996, Draft

[7] The DoD Test and Evaluation Process, June 13, 1994, 2.2.2.2.5

[8] The DoD Test and Evaluation Process, June 13, 1994, 2.2.2.2.4

3.2.10.6 Measurement Facilities (MF)

These are the measurement facility assets which are "visible" to the logical range.

These facilities are used to provide a specialized test environment and/or data collection capability. "Measurement facilities are used to quantify or measure parameters (such as thrust, radar cross section, and drag) of a test article in precise terms. Examples of such facilities are wind tunnels, radar cross section facilities, antennae pattern ranges, and engine thrust stands."[9] MF's may be ground-based laboratories or open air facilities (often located at or a part of OAR's).

> Component Measurement Test Events often involve the use of specialized capabilities to explore and evaluate advanced technologies and are usually the first test events performed during the development and/or buildup of the system. Examples include incoming parts inspection, thermal, acoustic and vibration cycling, power requirement and heat generation tests.[10]

[9] Simulation, Test, and Evaluation Process (STEP) Guidelines, December 2, 1996, Draft

[10] The DoD Test and Evaluation Process, June 13, 1994, 2.2.2.2.2

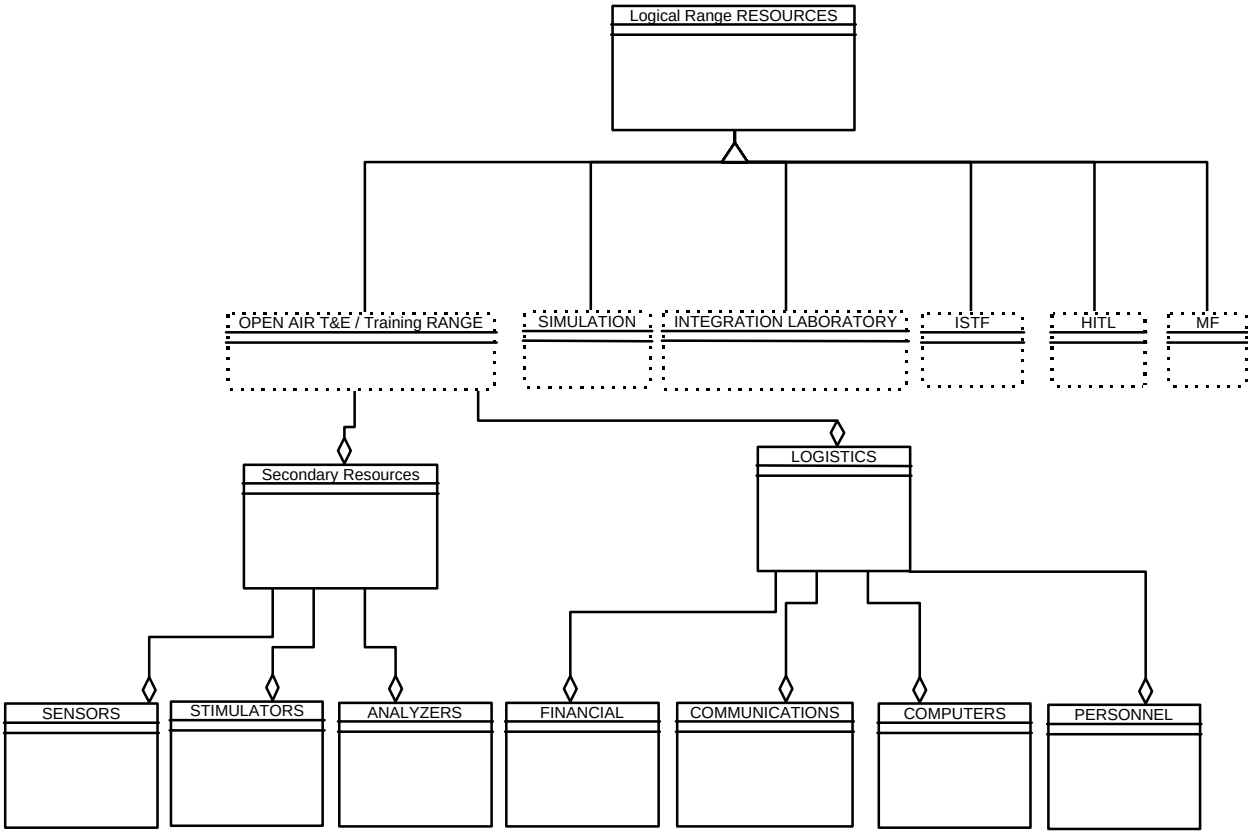

Figure 12. Logical Range Resource Class (Expanded)

3.2.11 Secondary Resources

Secondary Resources include facility or range specific assets that are required to support Primary Resources in the execution phase.

In Figure 12 the highlighted object classes are specific to the Open Air Range domain. It is anticipated that a similar class structure will apply to other facilities, but this has not been confirmed.

Secondary Resources are either specifically selected because of unique properties they posses, or they are required or defined by the selection of Mission Space elements. Secondary Resources in turn require or define Logistics resources. As an example, a Mission space requirement may include underwater tracking at the Underwater Tracking Range (UTR) at AFWTF, this in turn defines the hydrophone tracking array off of St. Croix, which in turn defines the Data Gathering and Processing System (DGPS) as a likely computer resource. A Customer could decide however, not to use the DGPS in favor of his own signal processing system which he could then connect to the Hydrophone Support Electronics (HSE) directly, and thus bypass the computer resources at the UTR

Further explanation of the object classes of Sensors, Stimulators and Analyzers are specific to the Open Air Range domain are given later in this document.

3.2.12 Logistics

The Logistics Class comprises those support asset classes required to actually execute a Logical Range Test/Training Exercise.

The selection of Secondary Resources (e.g., sensors, stimulators and analyzers) used to support the Logical Range Test/Training Exercise, imposes certain requirements and limitations on other necessary assets. These include Financial, Communications, Computers, and Personnel resources.

The Logistics class is conceptually different from the current representation of Secondary Resources, in that it is not domain specific, but rather specific to the whole of the TENA Technical Reference Architecture.

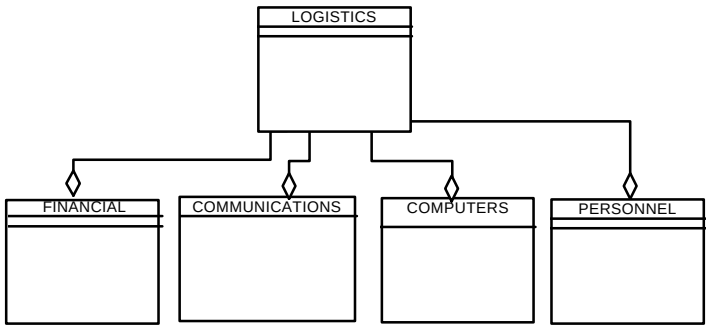

Figure 13. Logistics Class

The Logistics Class Structure, as shown in Figure 13, consists of the following sub-classes:

3.2.12.1 Financial

The Financial class represents those aspects of the planning and scheduling process that have cost implications to the Customer.

Each decision point in the Logical Range Business Process brings with it some financial data which may well direct the final choice of resources used to support the Logical Range Test/Training Exercise.

3.2.12.2 Communications Assets

Communications assets are used to move attribute information from one object to another, or to request an operation in an object be invoked.

The attribute descriptions in the object model help determine appropriate communication mechanisms. Attribute information should include characteristics like size, precision, update rate, and latency restrictions. The list of which attributes will be "published", and which are being "subscribed to" determines communication need paths. There must be sufficient information to determine if the communication needs of an exercise can be met. The communication class also needs operations to

set-up, monitor, and control the communication paths. These should include both static and dynamic services.

Details internal to the communication transport mechanism may not be visible to the Logical Range, especially for network communications via public carriers.

3.2.12.3 Computer Assets

Computer Assets represents the data processing capability which executes the operations on all of the objects in the Logical Range.

Every operation in the Logical Range must be assigned to some Computer Asset. Each computing assets has some inherent attributes like memory, processor power, secondary storage capacity, and access to communication channels. Planning of a Logical Range exercise includes ensuring there is at least one valid mapping of operations to computing assets which will meet all of the demands of the planned exercise. The mapping of operations to computer assets need not be static.

Computer assets can be pre-configured or optimized for certain kinds of operations such as MIS, Scientific Computation, and/or Data Acquisition operations. Computer assets may have subclasses like secondary storage, processor, display, and communication link. ****

3.2.12.4 Personnel

The Personnel class represents a map of those operations that a particular user/operator/observer is allowed to invoke for a Logical Range Test/Training Exercise.

Ranges tend to divide the responsibility for controlling / monitoring a test or exercise into predefined operational groups, i.e. range safety, test conductor, etc. The list of operations allowed for a specific kind of user can be saved and utilized for multiple tests, or a list can be customized for just one exercise.

Exercises which span multiple ranges (and into non-OAR domains) can run into operational confusion when, for example, the functions performed by a "Test Conductor" at Range A, differ from those at Range B. The TENA Object model contains a flexible mapping capability so there will be no ambiguity in what operations are allowed for different classes of user.

3.2 13 Secondary Resources

This section describes how the Secondary Resources Class (at the TRA level) is expanded for the specific domain of the Open Air Range. The Open Air Range specific subclasses are describe below.

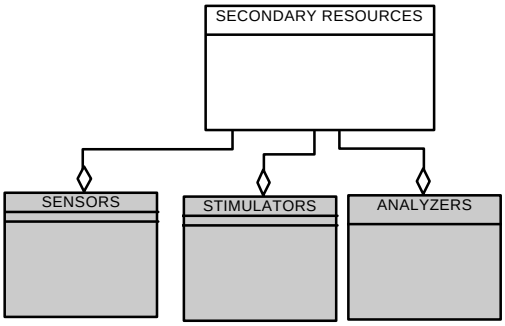

Figure 14. Secondary Resource Class

3.2.13.1 Sensors

The sensor class describes items which are used for the collection information from the environment or instrumented participants.

Sensors are most often used to determine the Time Space Position Information (TSPI) for a participant, standard range (Test / Exercise) time, environmental (weather, electromagnetic, etc.) characteristics and telemetry data from a participant. Sensor data can be correlated from multiple sensor sources, filtered from raw input data, or compared for "best solution" reporting.

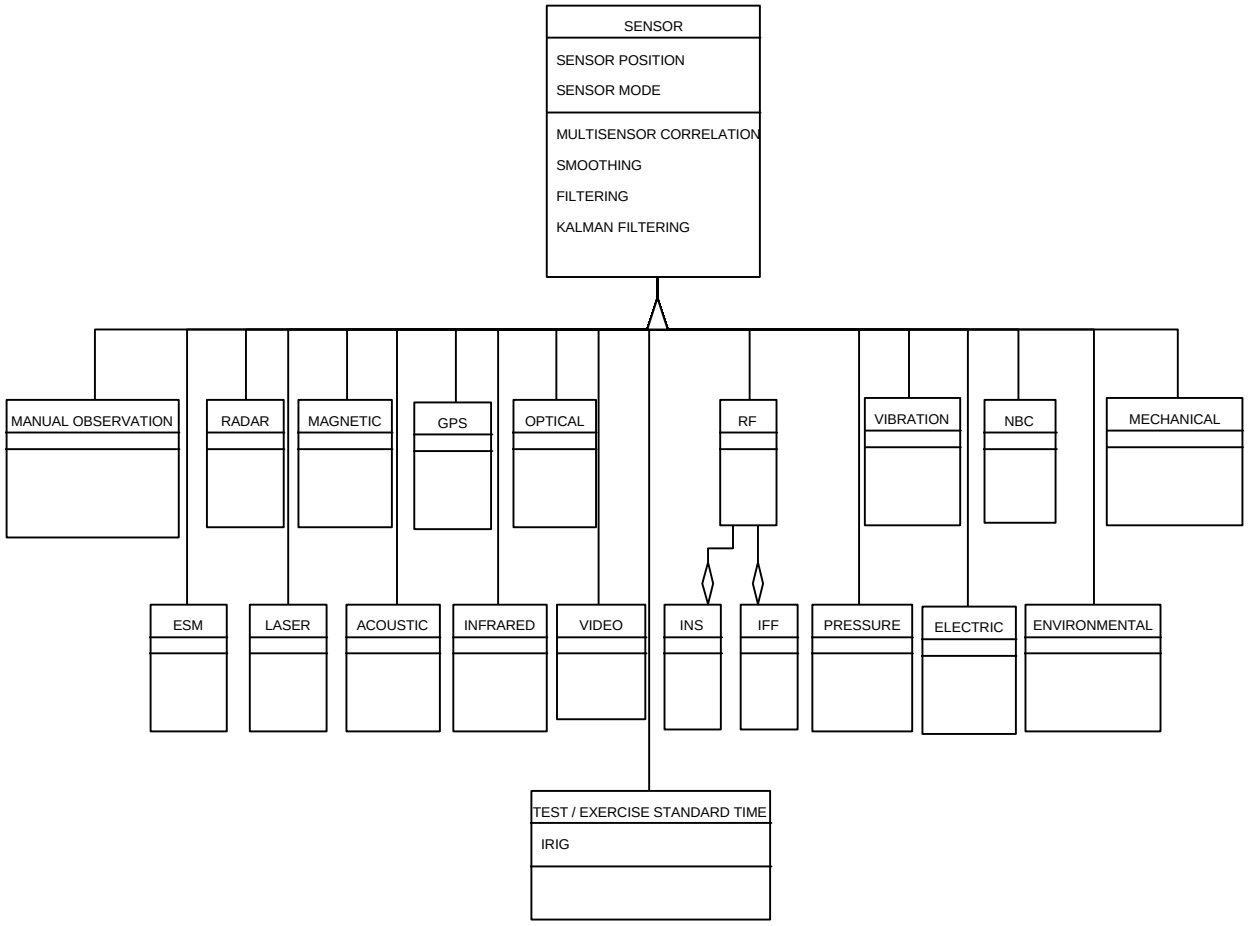

Figure 15. Sensor Class

3.2.13.1.1 Manual Observation

Manual Observations are those TSPI or data inputs made during or after real-time by human observers of the T&E/Training Logical Range Test/Training Exercise.

3.2.13.1.2 Electronic Support Measures (ESM)

ESM are the systems, or components of systems designed to detect and/or identify the presence of electronically radiated signals. ESM can be used for signature measurement of the Tactical Unit or System-Under-Test as well as provide platform performance.

3.2.13.1.3 Radar

Radar represents a sensor that is used to determine the distance and direction of objects by transmission and return of electromagnetic energy, which can be used for precision tracking and for safety control within the range area, and to determine such information as the Radar cross-section of a participant.

3.2.13.1.4 Laser

Laser represents a sensor designed to detect laser radiation.

3.2.13.1.5 Magnetic

Magnetic represents a sensor designed to detect magnetic fields, or the distortion of such fields.

3.2.13.1.6 Acoustic

Acoustic represents a sensor that uses sound/sound propagation designed to reflect and return objects of interest. This may include sonobuoys, sonar, and hydrophones.

3.2.13.1.7 Global Positioning System (GPS)

GPS is to used to provide highly accurate navigational position data by using a system of orbiting satellites. Time synchronization is also available from GPS.

3.2.13.1.8 Infra-Red

Infra-Red represents a sensor that provides the imagery produced as the result of sensing electromagnetic radiation emitted or reflected by the infra-red portion of the electromagnetic spectrum. The Infra-Red sensors are concerned mainly with heat radiated by an object, such as the hot plume of exhaust gas, aerodynamically generated heat, or even a person's heat. This information could be useful in object detection as well as evaluating the performance of certain platforms and in providing a signature measurement of the amount of heat generated by a unit.

3.2.13.1.9 Optical

Optical represents a sensor that pertains to sight or vision. This could include theodolites, binoculars, telescopes and television equipment. It may be more manual than electronic.

3.2.13.1.10 Video

Video is also a sensor that pertains to sight or vision, but it is distinct from the Optical Sensor in that it is entirely electronic.

3.2.13.1.11 Radio Frequency (RF)

RF Sensors are similar to Radar, except that they are passive receivers. RF is also used to transmit Inertial Navigation System (INS) data, as well as Identify Friend or Foe (IFF) mode information.

3.2.13.1.12 Test/Exercise Standard Time

Time synchronization is critical to the collection of real-time data. Most open air ranges receive time data as if it were sensed.

3.2.13.1.13 Pressure

Pressure represents a sensor that can be used to measure the exertion of force upon a surface by an object in contact with it. This sensor would be primarily used with telemetry systems.

3.2.13.1.14 Vibration

Vibration represents a sensor designed to detect or measure the vibrations of an object. This sensor would be primarily used with telemetry systems.

3.2.13.1.15 Electric

Electric represents those sensors or devices designed to measure or detect the presence of electrical fields/changes in an environment. This sensor would be primarily used with telemetry systems.

3.2.13.1.16 Nuclear Biological Chemical (NBC)

NBC sensors are designed to measure the levels and effects of NBC agents used during the conduct of a T&E/Training Logical Range Test/Training Exercise**.**

3.2.13.1.17 Mechanical

Mechanical data sensors are designed to measure stress, torque or other forces not included in the pressure or vibration classes

3.2.13.1.18 Environmental

Environmental Sensors measure the natural environment, such as wind direction and speed, temperature, humidity, atmospheric pressure, etc.

3.2.14 Stimulators

The stimulator class describes items which are used for influencing the environment and/or participants.

Stimulators influence the environment. The changes in the environment are then detected by range or participant sensors.

Electronic Counter Measures (ECM) is used for testing of electronic combat systems are an example of a class of stimulators. ECM stimulators are capable of providing a simulated hostile environment including radar, jamming, and various counter and counter-counter measures. ECM stimulators are also used for training.

Other stimulators correspond to the system or sensor that they are designed to stimulate.

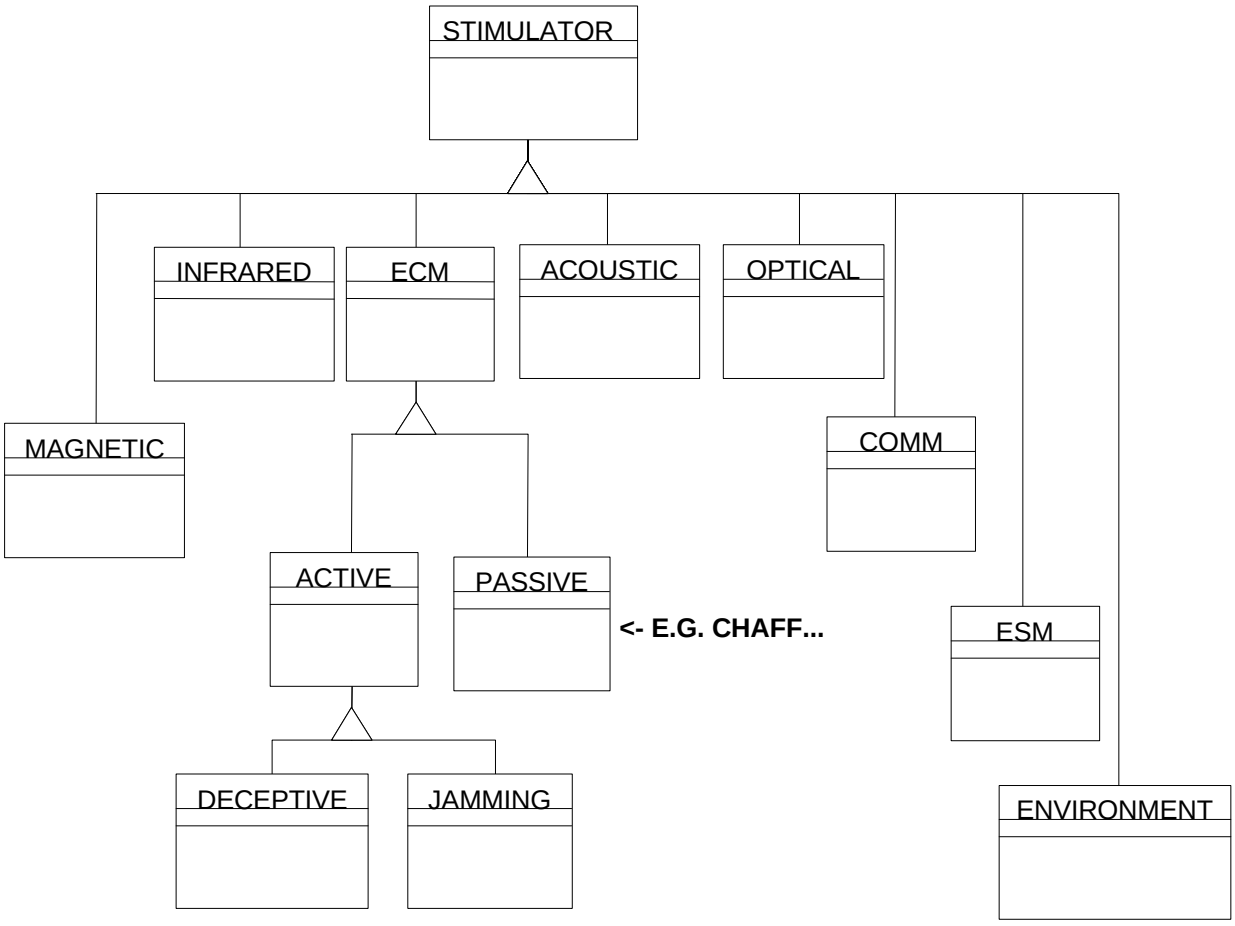

Figure 16. Stimulator Class

3.2.15 Analyzers

An analyzer uses information about a scenario to generate derived results / conclusions from that information. It may use archival data to assist in the derivation and/or conclusions.

3.3 Information Presenter Introduction

The Information Presenter Class (IPC) hierarchy provides a flexible (i.e., scaleable and extendible) framework for developing graphical user interfaces (GUIs). The IPC given borrows heavily from the Java Abstract Windowing Toolkit (AWT). The problems faced by the designers of the AWT are similar to those faced by TENA - namely, multiple independent platforms utilizing multiple independent and/or incompatible operating systems. By modeling the structure of the Information Presenter class after the AWT, TENA can leverage a reasonable COTS solution to a complex problem. However, the current AWT will not meet all of TENA's needs. Some modifications to the basic AWT structure are needed to address TENA specific concerns. Furthermore, limiting TENA to the AWT (and as a result Java) would be premature at this point in TENA's evolution.

From the perspective of the IPC, information is distinguished by its interpretation (at least in the digital sense). By distinguishing information in this manner, an IPC framework can be defined, as well as a set common of "information presenter" methods. The specific details regarding how and what information is interpreted becomes an implementation detail. A single presentation mechanism can be used to present all information regardless of media (e.g., CRT-based display or hardcopy).

The IPC is, in effect, independent of what is being presented. However, for information to be accurately presented some basic transactions must occur. Information interpretation must be consistent. Mechanisms to negotiate and fix the set(s) of methods/services for representing the information must be established. Namely,

- If the IPC has the appropriate information presentation capabilities, then these capabilities may be used,
- If the IPC doesn't have the appropriate information presentation capabilities, then these capabilities may be supplied, and
- If the IPC doesn't have the appropriate information presentation capabilities, and these capabilities are not supplied, the IPC cannot be used.

As an example of what would transpire during the negotiation and fixing of a method set, consider two possible means of displaying a raw video stream. First, the stream can be displayed on a Canvas using a paint method provided by the Canvas class. Second, the Canvas class may be subclassed and a new paint stream method supplied. (Note: a third, less deterministic, means exists in the form of dynamic method invocation, but that's out of the realm of this example and the current IPC).

Regarding the flexibility of the IPC as presented, consider the use of the TextComponet Class and its subclasses. Text is something that can be used in association with any displayable piece of the IPC component hierarchy, which in turn may present visual, aural, tactile information (i.e., text can be added to clarify visual, aural, tactile information without being a part of the actual information stream).

The complete IPC hierarchy is given in Figures 17-24. The following pages contain brief descriptions of each class in the IPC hierarchy. Also included are candidate sets of attributes and methods for each

class. These sets are not complete, but do offer a reasonable starting point from which the IPC can be further refined.

Classes given in the following description of the IPC do not specify any constructor methods. In a purely object oriented implementation of the IPC constructor methods would be required. The specification of the classes in a purely object oriented implementation would explicitly identify and describe the constructor methods.

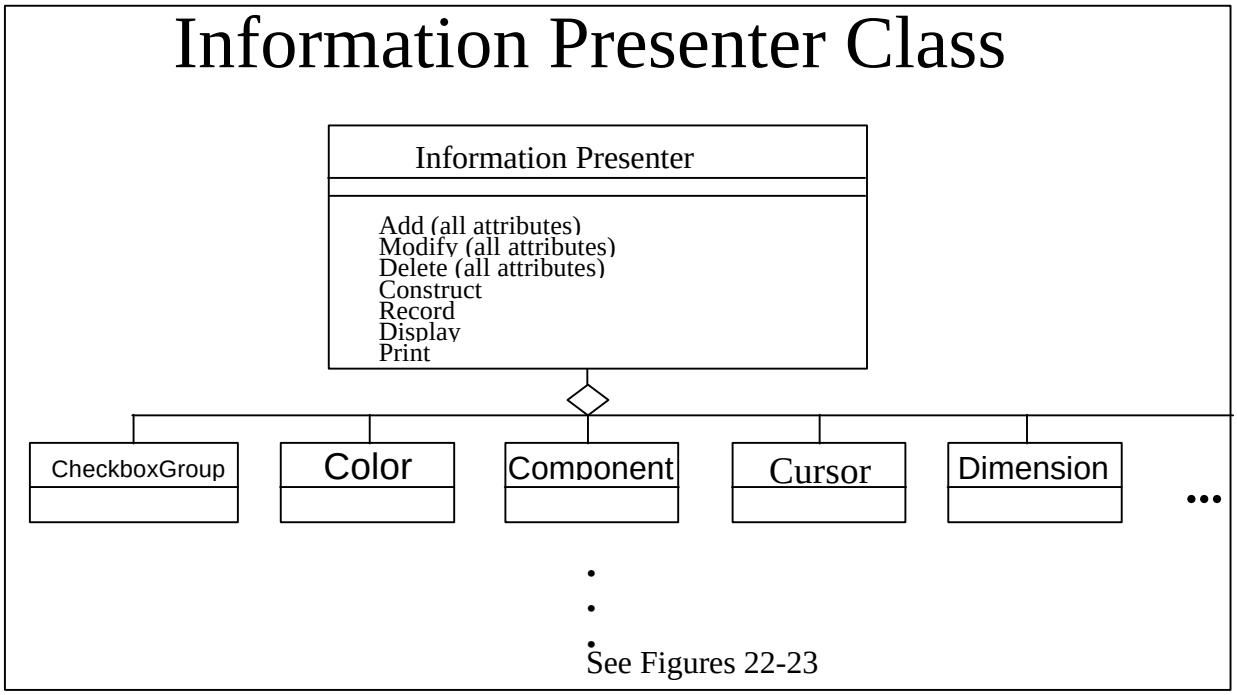

Figure 17. Information Presenter Class (1)

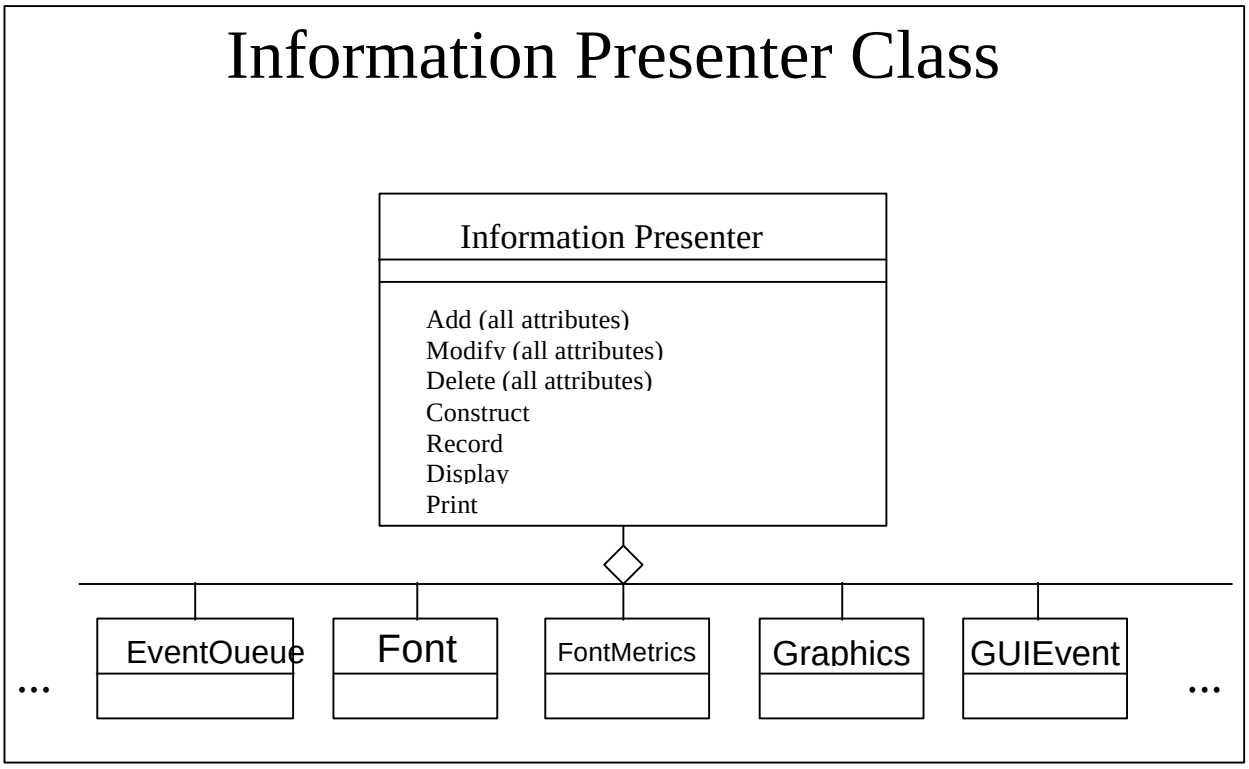

Figure 18. Information Presenter (2)

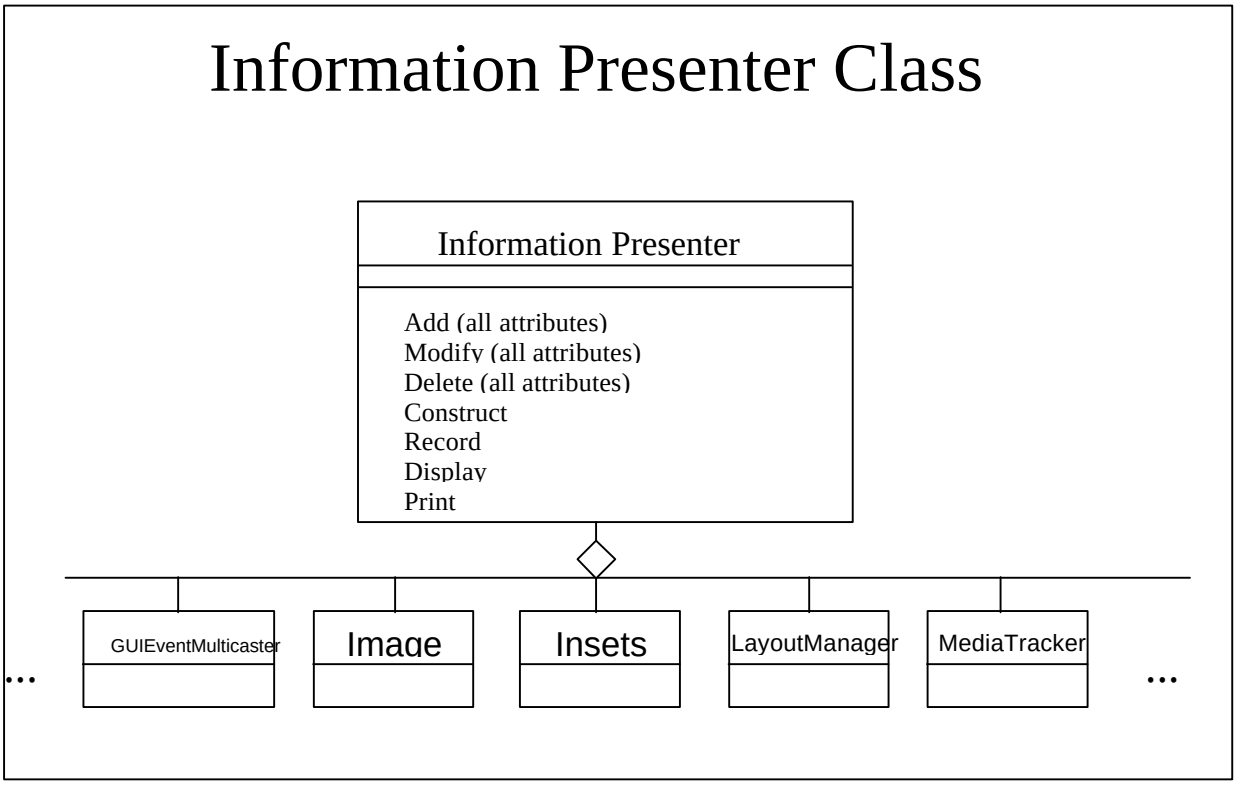

Figure 19. Information Presenter (3)

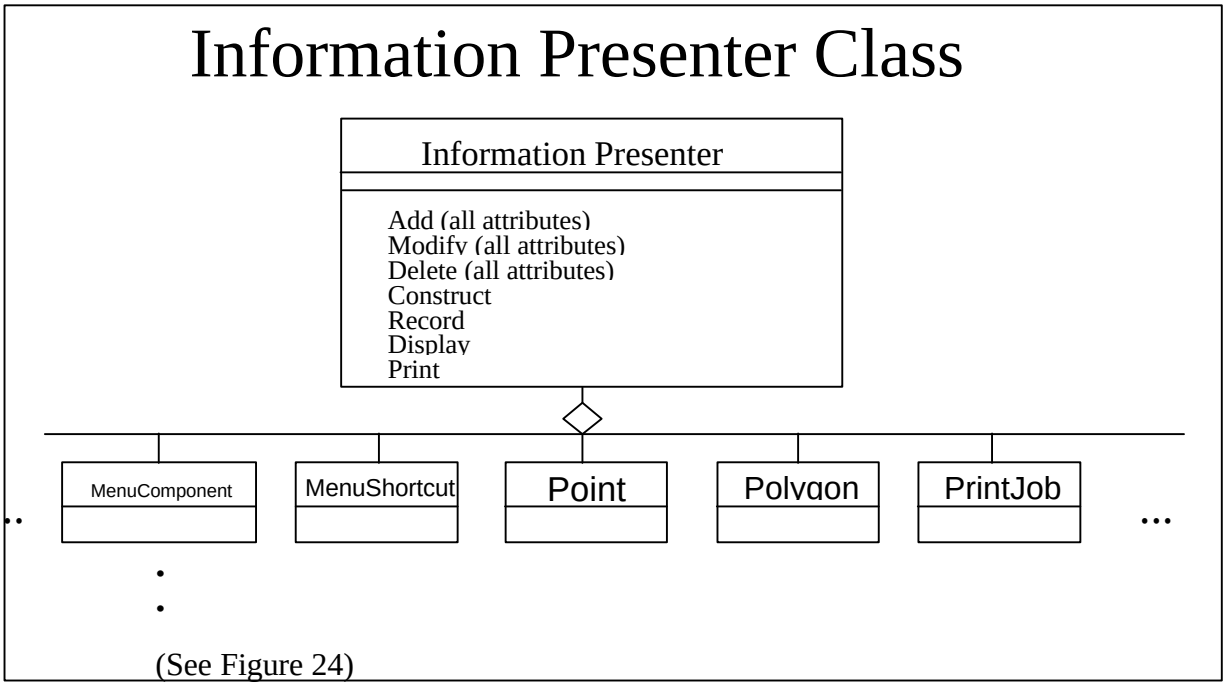

Figure 20. Information Presenter (4)

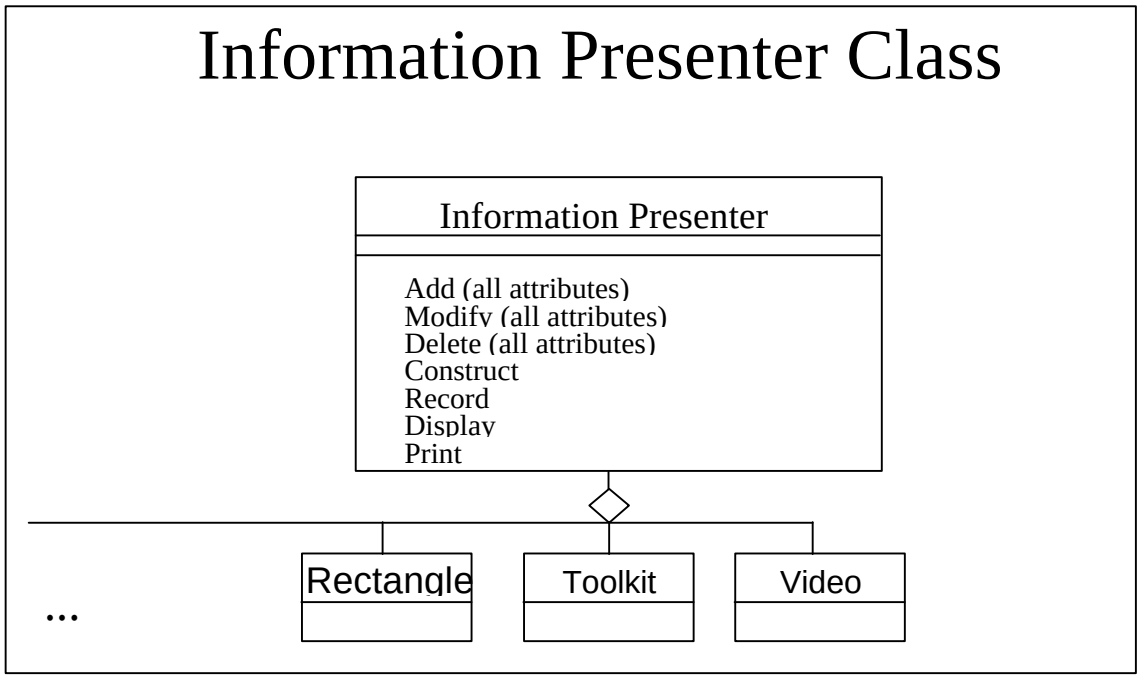

Figure 21. Information Presenter Class (5)

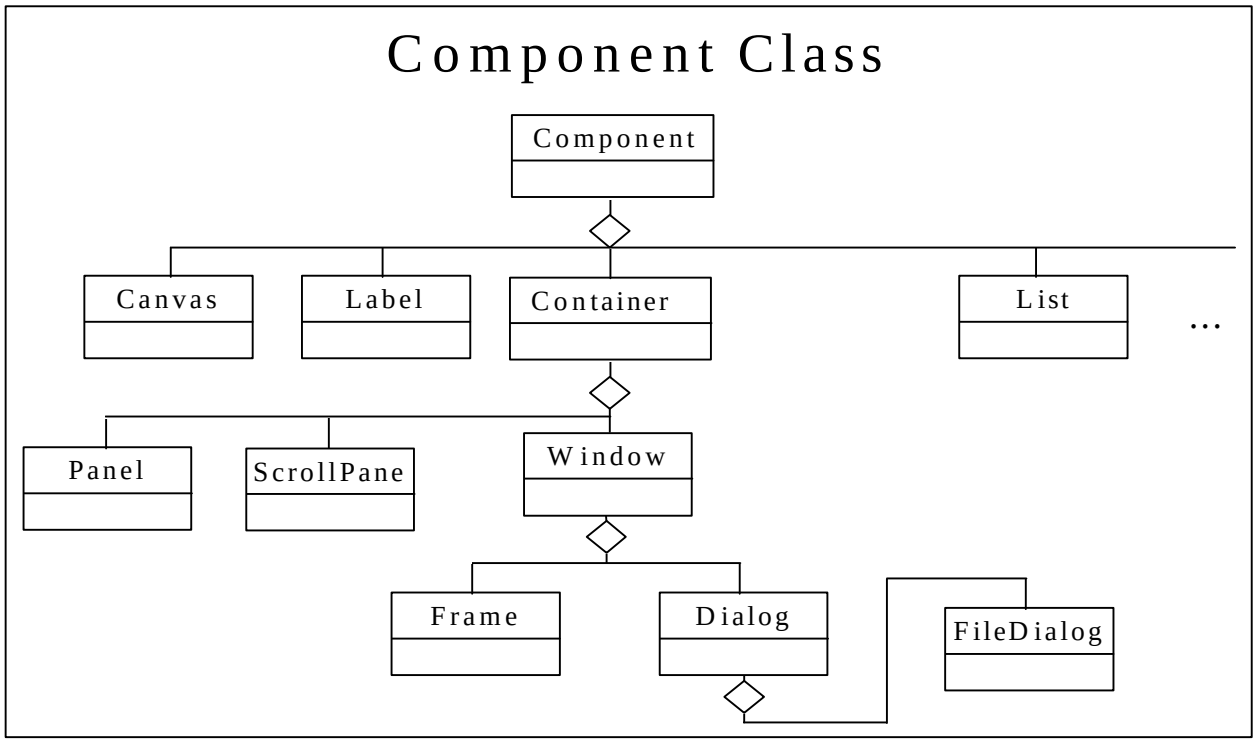

Figure 22. Component Class (1)

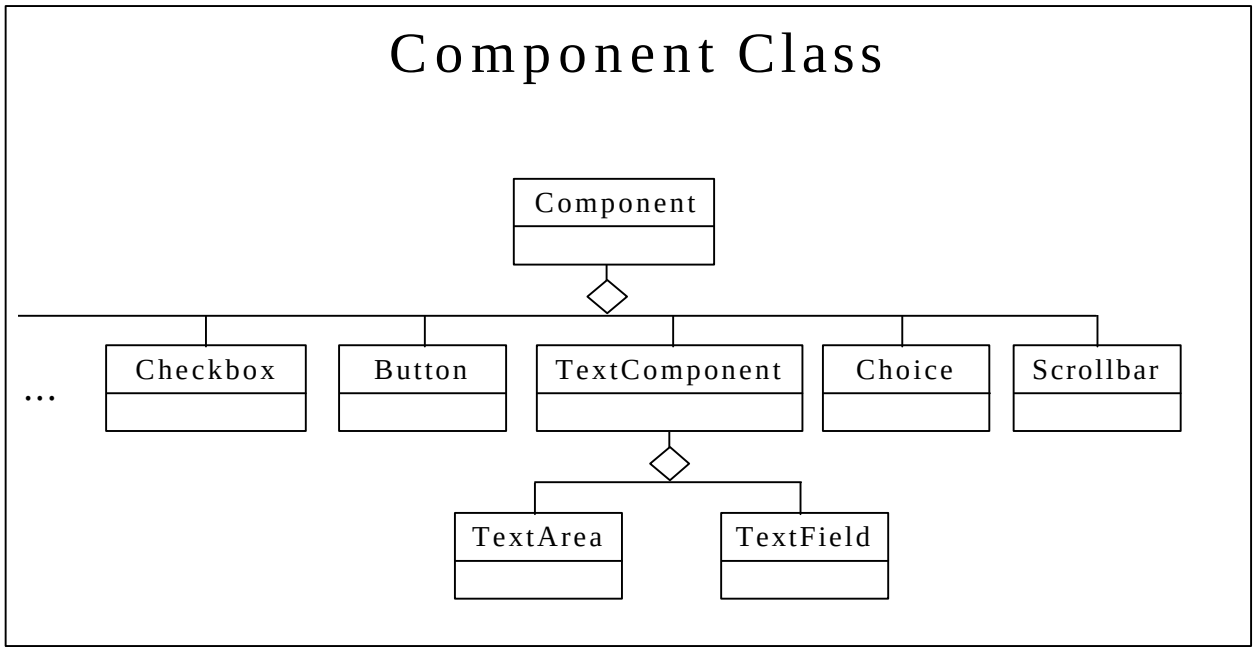

Figure 23. Component Class(2)

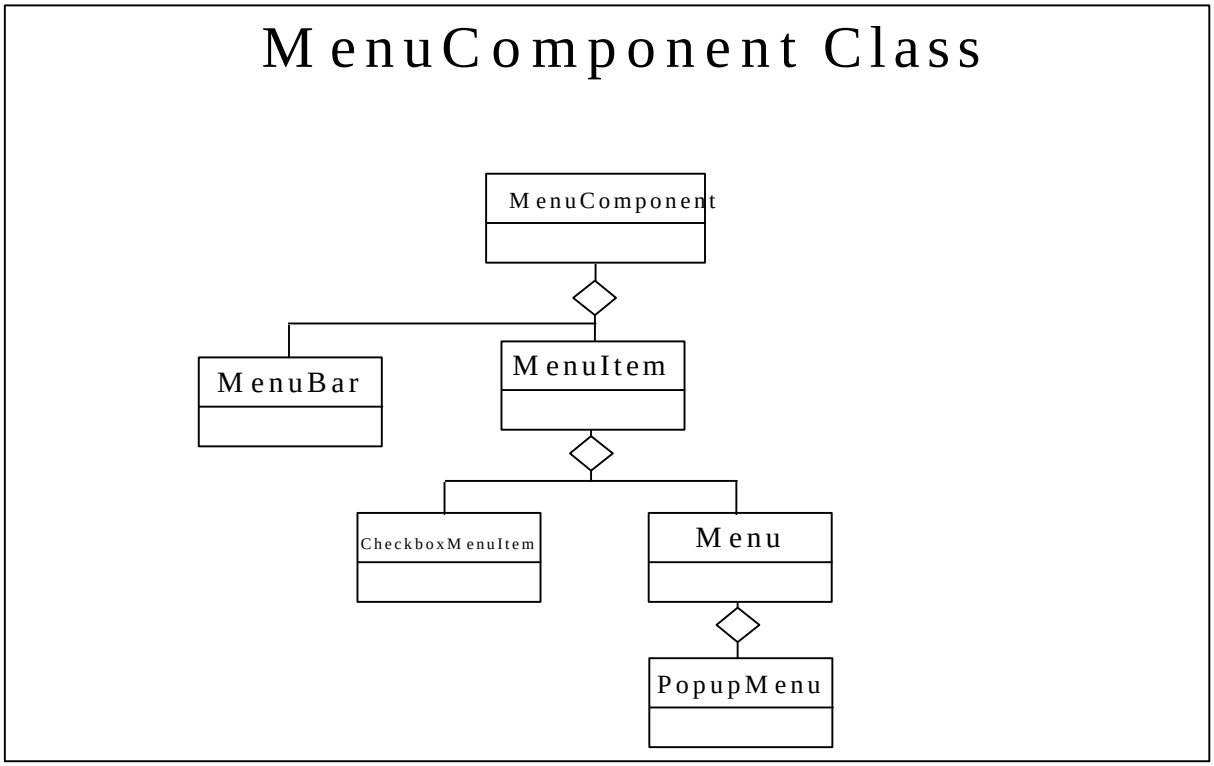

Figure 24. Menu Component Class

One note on nomenclature: There are several references to a Toolkit within the IPC. The IPC given relies heavily on an underlying windowing system (e.g., Windows or X-win). This reliance provides many advantages, chief of which is that fact that the IPC will not have to be built from scratch. The Toolkit is essentially middleware that binds the IPC components to their appropriate counterparts in the underlying windowing system. As such, the Toolkit is an implementation specific detail. However, the description of the IPC would be incomplete without at least mentioning the Toolkit.

3.4 Information Presenter Description

3.4.1 Class Name: Information Presenter

Description: The base class of all classes related to information presentation. The class also identifies utilities common to all information presentation subclasses.

Attributes:
Methods: Add (all attributes)
Modify (all attributes)
Delete (all attributes)
Construct
Display
Record
Print

3.4.2 Class Name: Checkbox Group

Description: The Checkbox Group class is used to group together a set of Checkbox components (see section 3.4.4.3).

Structure:
Specialization_Of: Information Presenter

Attributes:
Methods: getSelectedCheckbox()
Gets the current choice from this check box group.
setSelectedCheckbox(Checkbox)
Sets the currently selected check box in this group to be the specified check box.
toString()
Returns a string representation of this check box group, including the value of its current selection.

3.4.3 Class Name: Color

Description: This class encapsulates colors using the RGB format. In RGB format, the red, blue, and green components of a color are each represented by an integer in the range 0-255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.
Implementers of this class may find it useful to provide methods to convert between RGB (red, green, blue) and HSB (Hue Saturation Brightness) color formats.

Structure:
Specialization_Of: Information Presenter

Attributes:

black
: The color black.

blue
: The color blue.

cyan
: The color cyan.

darkGray
: The color dark gray.

gray
: The color gray.

green
: The color green.

lightGray
: The color light gray.

magenta
: The color magneta.

orange
: The color orange.

pink
: The color pink.

red
: The color red.

white
: The color white.

yellow
: The color yellow.

Methods:

brighter()
: Creates a brighter version of this color.

darker()
: Creates a darker version of this color.

decode(String)
: Converts a string to an integer and returns the specified color.

equals(Object)
: Determines whether another object is equal to this color.

getBlue()
: Gets the blue component of this color.

getColor(String)
: Finds a color in the system properties.

getColor(String, Color)
: Finds a color in the system properties.

getColor(String, int)
: Finds a color in the system properties.

getGreen()
: Gets the green component of this color.

getHSBColor(float, float, float)
: Creates a Color object based on values supplied for the HSB color model.

getRed()

Gets the red component of this color.

getRGB()

Gets the RGB value representing the color in the default RGB ColorModel.

HSBtoRGB(float, float, float)

Converts the components of a color, as specified by the HSB model, to an equivalent set of values for the RGB model.

RGBtoHSB(int, int, int, float[])

Converts the components of a color, as specified by the RGB model, to an equivalent set of values for hue, saturation, and brightness, the three components of the HSB model.

toString()

Creates a string that represents this color and indicates the values of its RGB components.

3.4.4 Class Name: Component

Description: A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Also, since the graphical representation can be displayed on the screen, it can be stored and printed as needed.

Structure:

Specialization_Of: Information Presenter

Attributes:

Methods: add(PopupMenu)

Adds the specified popup menu to the component.

checkImage(Image, ImageObserver)

Returns the status of the construction of a screen representation of the specified image.

checkImage(Image, int, int, ImageObserver)

Returns the status of the construction of a screen representation of the specified image.

contains(int, int)

Checks whether this component "contains" the specified point, where x and y are defined to be relative to the coordinate system of this component.

contains(Point)

Checks whether this component "contains" the specified point, where the point's x and y coordinates are defined to be relative to the coordinate system of this component.

createImage(ImageProducer)

Creates an image from the specified image producer.

createImage(int, int)

Creates an off-screen drawable image to be used for double buffering.

doLayout()

Prompts the layout manager to lay out this component.

getAlignmentX()

Returns the alignment along the x axis.

getAlignmentY()

Returns the alignment along the y axis.

getBackground()

Gets the background color of this component.

getBounds()

Gets the bounds of this component in the form of a Rectangle object.

getColorModel()

Gets the instance of ColorModel used to display the component on the output device.

getComponentAt(int, int)

Determines if this component or one of its immediate subcomponents contains the (x, y) location, and if so, returns the containing component.

getComponentAt(Point)

Returns the component or subcomponent that contains the specified point.

getCursor()

Gets the cursor set on this component.

getFont()

Gets the font of this component.

getForeground()

Gets the foreground color of this component.

getGraphics()

Creates a graphics context for this component.

getLocale()

Gets the locale of this component.

getLocation()

Gets the location of this component in the form of a point specifying the component's top-left corner.

getLocationOnScreen()

Gets the location of this component in the form of a point specifying the component's top-left corner in the screen's coordinate space.

getMaximumSize()

Gets the maximum size of this component.

getMinimumSize()

Gets the mininimum size of this component.

getName()

Gets the name of the component.

getParent()

Gets the parent of this component.

getPreferredSize()

Gets the preferred size of this component.

getSize()

Returns the size of this component in the form of a Dimension object.

getToolkit()

Gets the toolkit of this component.

getTreeLock()

Gets this component's locking object (the object that owns the thread sychronization monitor) for AWT component-tree and layout operations.

imageUpdate(Image, int, int, int, int, int)

Repaints the component when the image has changed.

invalidate()

Invalidates this component.

isEnabled()

Determines whether this component is enabled.

isFocusTraversable()

Returns the value of a flag that indicates whether this component can be traversed using Tab or Shift-Tab keyboard focus traversal.

isShowing()

Determines whether this component is showing on screen.

isValid()

Determines whether this component is valid.

isVisible()

Determines whether this component is visible.

list()

Prints a listing of this component to the standard system output stream System.out.

list(PrintStream)
Prints a listing of this component to the specified output stream.
list(PrintStream, int)
Prints out a list, starting at the specified indention, to the specified print stream.
list(PrintWriter)
Prints a listing to the specified print writer.
list(PrintWriter, int)
Prints out a list, starting at the specified indention, to the specified print writer.
paint(Graphics)
Paints this component.
paintAll(Graphics)
Paints this component and all of its subcomponents.
paramString()
Returns the parameter string representing the state of this component.
prepareImage(Image, ImageObserver)
Prepares an image for rendering on this component.
prepareImage(Image, int, int, ImageObserver)
Prepares an image for rendering on this component at the specified width and height.
print(Graphics)
Prints this component.
printAll(Graphics)
Prints this component and all of its subcomponents.
processComponentEvent(ComponentEvent)
Processes component events occurring on this component by dispatching them to any appropriate objects.
processEvent(AWTEvent)
Processes events occurring on this component.
processFocusEvent(FocusEvent)
Processes focus events occurring on this component by dispatching them to any appropriate objects.
processKeyEvent(KeyEvent)
Processes key events occurring on this component by dispatching them to any registered objects.
processMouseEvent(MouseEvent)
Processes mouse events occurring on this component by dispatching them to any appropriate objects.
processMouseMotionEvent(MouseEvent)
Processes mouse motion events occurring on this component by dispatching them to any appropriate objects.
remove(MenuComponent)
Removes the specified popup menu from the component.
removeNotify()
Notifies this component that it has been removed from its container and if a peers exists, it destroys it.
repaint()
Repaints this component.
repaint(int, int, int, int)
Repaints the specified rectangle of this component.
repaint(long)
Repaints the component.
repaint(long, int, int, int, int)
Repaints the specified rectangle of this component within tm milliseconds.
requestFocus()
Requests that this component get the input focus.

setBackground(Color)
Sets the background color of this component.
setBounds(int, int, int, int)
Moves and resizes this component.
setBounds(Rectangle)
Moves and resizes this component to conform to the new bounding rectangle r.
setCursor(Cursor)
Set the cursor image to a predefined cursor.
setEnabled(boolean)
Enables or disables this component, depending on the value of the parameter b.
setFont(Font)
Sets the font of this component.
setForeground(Color)
Sets the foreground color of this component.
setLocale(Locale)
Sets the locale of this component.
setLocation(int, int)
Moves this component to a new location.
setLocation(Point)
Moves this component to a new location.
setName(String)
Sets the name of the component to the specified string.
setSize(Dimension)
Resizes this component so that it has width d.width and height d.height.
setSize(int, int)
Resizes this component so that it has width width and height.
setVisible(boolean)
Shows or hides this component depending on the value of parameter b.
toString()
Returns a string representation of this component and its values.
transferFocus()
Transfers the focus to the next component.
update(Graphics)
Updates this component.
validate()
Ensures that this component has a valid layout.

3.4.4.1 Class Name: Button

Description: This class creates a labeled button. An application can cause some action to happen when the button is pushed.

Structure:
Specialization_Of: Component

Attributes:

Methods: addNotify()
Creates the peer of the button.
getActionCommand()
Returns the command name of the action event fired by this button.
getLabel()
Gets the label of this button.

paramString()
Returns the parameter string representing the state of this button.
processActionEvent(ActionEvent)
Processes action events occurring on this button by dispatching them to any appropriate objects.
processEvent(AWTEvent)
Processes events on this button.
setActionCommand(String)
Sets the command name for the action event fired by this button.
setLabel(String)
Sets the button's label to be the specified string.

3.4.4.2 Class Name: Canvas

Description: A Canvas component represents a blank rectangular area of the screen onto which an application can draw or from which an application can trap input events from the user. An application must subclass the Canvas class in order to get useful functionality such as creating a custom component. The paint method must be overridden in order to perform custom graphics on the canvas.

Structure:
Specialization_Of: Component

Attributes:

Methods: addNotify()
Creates the peer of the canvas.
paint(Graphics)
This method is called to repaint this canvas.

3.4.4.3 Class Name: Checkbox

Description: A check box is a graphical component that can be in either an "on" (true) or "off" (false) state. Clicking on a check box changes its state from "on" to "off," or from "off" to "on.

Structure:
Specialization_Of: Component

Attributes:

Methods: addNotify()
Creates the peer of the Checkbox.
getCheckboxGroup()
Determines this check box's group.
getLabel()
Gets the label of this check box.
getSelectedObjects()
Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected.
getState()
Determines whether this check box is in the "on" or "off" state.
paramString()
Returns the parameter string representing the state of this check box.
processEvent(AWTEvent)

Processes events on this check box.

processItemEvent(ItemEvent)

Processes item events occurring on this check box by dispatching them to any appropriate objects.

setCheckboxGroup(CheckboxGroup)

Sets this check box's group to be the specified check box group.

setLabel(String)

Sets this check box's label to be the string argument.

setState(boolean)

Sets the state of this check box to the specified state.

3.4.4.4 Class Name: Choice

Description: The Choice class presents a pop-up menu of choices. The current choice is displayed as the title of the menu.

Structure:

Specialization_Of: Component

Attributes:

Methods:

add(String)

Adds an item to this Choice menu.

addItem(String)

Adds an item to this Choice.

addNotify()

Creates the Choice's peer.

getItem(int)

Gets the string at the specified index in this Choice menu.

getItemCount()

Returns the number of items in this Choice menu.

getSelectedIndex()

Returns the index of the currently selected item.

getSelectedItem()

Gets a representation of the current choice as a string.

getSelectedObjects()

Returns an array (length 1) containing the currently selected item.

insert(String, int)

Inserts the item into this choice at the specified position.

paramString()

Returns the parameter string representing the state of this choice menu.

processEvent(AWTEvent)

Processes events on this choice.

processItemEvent(ItemEvent)

Processes item events occurring on this Choice menu by dispatching them to any appropriate objects.

remove(int)

Removes an item from the choice menu at the specified position.

remove(String)

Remove the first occurrence of item from the Choice menu.

removeAll()

Removes all items from the choice menu.

select(int)

Sets the selected item in this Choice menu to be the item at the specified position.

select(String)

Sets the selected item in this Choice menu to be the item whose name is equal to the specified string.

3.4.4.5 Class Name: Container

Description: A container object is a component that can contain other components.

Structure:

Specialization_Of: Component

Attributes:

Methods:

add(Component)

Adds the specified component to the end of this container.

add(Component, int)

Adds the specified component to this container at the given position.

add(Component, Object)

Adds the specified component to the end of this container.

add(Component, Object, int)

Adds the specified component to this container with the specified constraints at the specified index.

add(String, Component)

Adds the specified component to this container.

addImpl(Component, Object, int)

Adds the specified component to this container at the specified index.

addNotify()

Notifies the container to create a peer.

doLayout()

Causes this container to lay out its components.

getAlignmentX()

Returns the alignment along the x axis.

getAlignmentY()

Returns the alignment along the y axis.

getComponent(int)

Gets the nth component in this container.

getComponentAt(int, int)

Locates the component that contains the x,y position.

getComponentAt(Point)

Gets the component that contains the specified point.

getComponentCount()

Gets the number of components in this panel.

getComponents()

Gets all the components in this container.

getInsets()

Determines the insets of this container, which indicate the size of the container's border.

getLayout()

Gets the layout manager for this container.

getMaximumSize()

Returns the maximum size of this container.

getMinimumSize()

Returns the minimum size of this container.

getPreferredSize()

Returns the preferred size of this container.

invalidate()

Invalidates the container.

isAncestorOf(Component)

Checks if the component is contained in the component hierarchy of this container.

list(PrintStream, int)

Prints a listing of this container to the specified output stream.

list(PrintWriter, int)

Prints out a list, starting at the specified indention, to the specified print writer.

paint(Graphics)

Paints the container.

paintComponents(Graphics)

Paints each of the components in this container.

paramString()

Returns the parameter string representing the state of this container.

print(Graphics)

Prints the container.

printComponents(Graphics)

Prints each of the components in this container.

processContainerEvent(ContainerEvent)

Processes container events occurring on this container by dispatching them to any appropriate objects.

processEvent(AWTEvent)

Processes events on this container.

remove(Component)

Removes the specified component from this container.

remove(int)

Removes the component, specified by index, from this container.

removeAll()

Removes all the components from this container.

removeNotify()

Notifies this container and all of its subcomponents to remove their peers.

setLayout(LayoutManager)

Sets the layout manager for this container.

validate()

Validates this container and all of its subcomponents.

validateTree()

Recursively descends the container tree and recomputes the layout for any subtrees marked as needing it (those marked as invalid).

3.4.4.5.1 Class Name: Panel

Description: Panel is the simplest container class. A panel provides space in which an application can attach any other component, including other panels.

Structure:

Specialization_Of: Container

Attributes:

Methods: addNotify()
Creates the Panel's peer.

3.4.4.5.2 Class Name: Scrollpane

Description: A container class which implements automatic horizontal and/or vertical scrolling for a single child component. The display policy for the scrollbars can be set to:

1. 1. as needed: scrollbars created and shown only when needed by scrollpane;
2. 2. always: scrollbars created and always shown by the scrollpane;
3. 3. never: scrollbars never created or shown by the scrollpane.

Structure:
Specialization_Of: Container

Attributes: SCROLLBARS_ALWAYS
Specifies that horizontal/vertical scrollbars should always be shown regardless of the respective sizes of the scrollpane and child.
SCROLLBARS_AS_NEEDED
Specifies that horizontal/vertical scrollbar should be shown only when the size of the child exceeds the size of the scrollpane in the horizontal/vertical dimension.
SCROLLBARS_NEVER
Specifies that horizontal/vertical scrollbars should never be shown regardless of the respective sizes of the scrollpane and child.

Methods: addImpl(Component, Object, int)
Adds the specified component to this scroll pane container.
addNotify()
Creates the scroll pane's peer.
doLayout()
Lays out this container by resizing its child to its preferred size.
getHAdjustable()
Returns the Adjustable object which represents the state of the horizontal scrollbar.
getHScrollbarHeight()
Returns the height that would be occupied by a horizontal scrollbar, which is independent of whether it is currently displayed by the scroll pane or not.
getScrollbarDisplayPolicy()
Returns the display policy for the scrollbars.
getScrollPosition()
Returns the current x,y position within the child which is displayed at the 0,0 location of the scrolled panel's view port.
getVAdjustable()
Returns the Adjustable object which represents the state of the vertical scrollbar.
getViewportSize()
Returns the current size of the scroll pane's view port.
getVScrollbarWidth()
Returns the width that would be occupied by a vertical scrollbar, which is independent of whether it is currently displayed by the scroll pane or not.
paramString()
Returns the parameter string representing the state of this container.
printComponents(Graphics)
Prints the component in this scroll pane.

setLayout(LayoutManager)
Sets the layout manager for this container.
setScrollPosition(int, int)
Scrolls to the specified position within the child component.
setScrollPosition(Point)
Scrolls to the specified position within the child component.

3.4.4.5.3 Class Name: Window

Description: A Window object is a top-level window with no borders and no menubar. It could be used to implement a pop-up menu. A Window object blocks input to other application windows when it is shown.

Structure:

Specialization_Of: Container

Attributes:

Methods: addNotify()
Creates the Window's peer.
dispose()
Disposes of this window.
getFocusOwner()
Returns the child component of this Window which has focus if and only if this Window is active.
getLocale()
Gets the Locale object that is associated with this window, if the locale has been set.
getToolkit()
Returns the toolkit of this frame.
getWarningString()
Gets the warning string that is displayed with this window.
isShowing()
Checks if this Window is showing on screen.
pack()
Causes subcomponents of this window to be laid out at their preferred size.
processEvent(AWTEvent)
Processes events on this window.
processWindowEvent(WindowEvent)
Processes window events occurring on this window by dispatching them to the appropriate objects.
show()
Shows this window, and brings it to the front.
toBack()
Sends this window to the back.
toFront()
Brings this window to the front.

3.4.4.5.3.1 Class Name: Frame

Description: A Frame is a top-level window with a title and a border.

Structure:

Specialization_Of: Window

Attributes:

Methods: addNotify()
Creates the Frame's peer.
dispose()
Disposes of the Frame.
getIconImage()
Gets the icon image for this frame.
getMenuBar()
Gets the menu bar for this frame.
getTitle()
Gets the title of the frame.
isResizable()
Indicates whether this frame is resizable.
paramString()
Returns the parameter String of this Frame.
remove(MenuComponent)
Removes the specified menu bar from this frame.
setIconImage(Image)
Sets the image to display when this frame is iconized.
setMenuBar(MenuBar)
Sets the menu bar for this frame to the specified menu bar.
setResizable(boolean)
Sets the resizable flag, which determines whether this frame is resizable.
setTitle(String)
Sets the title for this frame to the specified title.

3.4.4.5.3.2 Class Name: Dialog

Description: A class that produces a dialog - a window that takes input from the user.

Structure:
Specialization_Of: Window

Attributes:

Methods: addNotify()
Creates the dialog's peer.
getTitle()
Gets the title of the dialog.
isModal()
Indicates whether the dialog is modal.
isResizable()
Indicates whether this dialog window is resizable.
paramString()
Returns the parameter string representing the state of this dialog window.
setModal(boolean)
Specifies whether this dialog is modal.
setResizable(boolean)
Sets the resizable flag.
setTitle(String)

Sets the title of the Dialog.

show()

Shows the dialog.

3.4.4.5.3.2.1 Class Name: FileDialog

Description: The FileDialog class displays a dialog window from which the user can select a file.

Structure:

Specialization_Of: Dialog

Attributes: LOAD

This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.

SAVE

This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.

Methods: addNotify()

Creates the file dialog's peer.

getDirectory()

Gets the directory of this file dialog.

getFile()

Gets the selected file of this file dialog.

getFilenameFilter()

Determines this file dialog's filename filter.

getMode()

Indicates whether this file dialog box is for loading from a file or for saving to a file.

paramString()

Returns the parameter string representing the state of this file dialog window.

setDirectory(String)

Sets the directory of this file dialog window to be the specified directory.

setFile(String)

Sets the selected file for this file dialog window to be the specified file.

setFilenameFilter(FilenameFilter)

Sets the filename filter for this file dialog window to the specified filter.

setMode(int)

Sets the mode of the file dialog.

3.4.4.6 Class Name: Label

Description: A Label object is a component for placing text in a container. A label displays a single line of read-only text. The text can be changed by the application, but a user cannot edit it directly.

Structure:

Specialization_Of: Component

Attributes:

Methods: addNotify()

Creates the peer for this label.

getAlignment()

Gets the current alignment of this label.

getText()

Gets the text of this label.

paramString()

Returns the parameter string representing the state of this label.

setAlignment(int)

Sets the alignment for this label to the specified alignment.

setText(String)

Sets the text for this label to the specified text.

3.4.4.7 Class Name: List

Description: The List component presents the user with a scrolling list of text items. The list can be set up so that the user can choose either one item or multiple items.

Structure:

Specialization_Of: Component

Attributes:

Methods:

add(String)

Adds the specified item to the end of scrolling list.

add(String, int)

Adds the specified item to the end of the scrolling list.

addNotify()

Creates the peer for the list.

deselect(int)

Deselects the item at the specified index.

getItem(int)

Gets the item associated with the specified index.

getItemCount()

Gets the number of items in the list.

getItems()

Gets the items in the list.

getMinimumSize()

Determines the minimum size of this scrolling list.

getMinimumSize(int)

Gets the minumum dimensions for a list with the specified number of rows.

getPreferredSize()

Gets the preferred size of this scrolling list.

getPreferredSize(int)

Gets the preferred dimensions for a list with the specified number of rows.

getRows()

Get the number of visible lines in this list.

getSelectedIndex()

Gets the index of the selected item on the list,

getSelectedIndexes()

Gets the selected indexes on the list.

getSelectedItem()

Get the selected item on this scrolling list.

getSelectedItems()

Get the selected items on this scrolling list.

getSelectedObjects()

Returns the selected items on the list in an array of Objects.

getVisibleIndex()

Gets the index of the item that was last made visible by the method makeVisible.

isIndexSelected(int)

Determines if the specified item in this scrolling list is selected.

isMultipleMode()

Determines whether this list allows multiple selections.

makeVisible(int)

Makes the item at the specified index visible.

paramString()

Returns the parameter string representing the state of this scrolling list.

processActionEvent(ActionEvent)

Processes action events occurring on this component by dispatching them to the appropriate registered objects.

processEvent(AWTEvent)

Processes events on this scrolling list.

processItemEvent(ItemEvent)

Processes item events occurring on this list by dispatching them to the appropriate registered objects.

remove(int)

Remove the item at the specified position from this scrolling list.

remove(String)

Removes the first occurrence of an item from the list.

removeAll()

Removes all items from this list.

removeNotify()

Removes the peer for this list.

replaceItem(String, int)

Replaces the item at the specified index in the scrolling list with the new string.

select(int)

Selects the item at the specified index in the scrolling list.

setMultipleMode(boolean)

Sets the flag that determines whether this list allows multiple selections.

3.4.4.8 Class Name: Scrollbar

Description: The Scrollbar class embodies a scroll bar, a familiar user-interface object. A scroll bar provides a convenient means for allowing a user to select from a range of values.

Structure:

Specialization_Of: Component

Attributes:

Methods: addNotify()

Creates the Scrollbar's peer.

getBlockIncrement()

Gets the block increment of this scroll bar.

getMaximum()

Gets the maximum value of this scroll bar.

getMinimum()

Gets the minimum value of this scroll bar.

getOrientation()

Determines the orientation of this scroll bar.

getUnitIncrement()

Gets the unit increment for this scrollbar.

getValue()

Gets the current value of this scroll bar.

getVisibleAmount()

Gets the visible amount of this scroll bar.

paramString()

Returns the parameter string representing the state of this scroll bar.

processAdjustmentEvent(AdjustmentEvent)

Processes adjustment events occurring on this scrollbar by dispatching them to the appropriate registered objects.

processEvent(AWTEvent)

Processes events on this scroll bar.

setBlockIncrement(int)

Sets the block increment for this scroll bar.

setMaximum(int)

Sets the maximum value of this scroll bar.

setMinimum(int)

Sets the minimum value of this scroll bar.

setOrientation(int)

Sets the orientation for this scroll bar.

setUnitIncrement(int)

Sets the unit increment for this scroll bar.

setValue(int)

Sets the value of this scroll bar to the specified value.

setValues(int, int, int, int)

Sets the values of four properties for this scroll bar.

setVisibleAmount(int)

Sets the visible amount of this scroll bar.

3.4.4.9 Class Name: TextComponent

Description: The TextComponent class is the superclass of any component that allows the editing of some text.

Structure:

Specialization_Of: Component

Attributes:

Methods:

getCaretPosition()

Gets the position of the text insertion caret for this text component.

getSelectedText()

Gets the selected text from the text that is presented by this text component.

getSelectionEnd()

Gets the end position of the selected text in this text component.

getSelectionStart()

Gets the start position of the selected text in this text component.

getText()

Gets the text that is presented by this text component.

isEditable()

Indicates whether or not this text component is editable.

paramString()

Returns the parameter string representing the state of this text component.

processEvent(AWTEvent)
Processes events on this textcomponent.
processTextEvent(TextEvent)
Processes text events occurring on this text component by dispatching them to any appropriate objects.
removeNotify()
Removes the TextComponent's peer.
select(int, int)
Selects the text between the specified start and end positions.
selectAll()
Selects all the text in this text component.
setCaretPosition(int)
Sets the position of the text insertion caret for this text component.
setEditable(boolean)
Sets the flag that determines whether or not this text component is editable.
setSelectionEnd(int)
Sets the selection end for this text component to the specified position.
setSelectionStart(int)
Sets the selection start for this text component to the specified position.
setText(String)
Sets the text that is presented by this text component to be the specified text.

3.4.4.9.1 Class Name: TextArea

Description: A TextArea object is a multi-line region that displays text. It can be set to allow editing or to be read-only.

Structure:
Specialization_Of: TextComponent

Attributes: SCROLLBARS_BOTH
Create and display both vertical and horizontal scrollbars.
SCROLLBARS_HORIZONTAL_ONLY
Create and display horizontal scrollbar only.
SCROLLBARS_NONE
Do not create or display any scrollbars for the text area.
SCROLLBARS_VERTICAL_ONLY
Create and display vertical scrollbar only.

Methods: addNotify()
Creates the TextArea's peer.
append(String)
Appends the given text to the text area's current text.
getColumns()
Gets the number of columns in this text area.
getMinimumSize()
Determines the minimum size of this text area.
getMinimumSize(int, int)
Determines the minimum size of a text area with the specified number of rows and columns.
getPreferredSize()
Determines the preferred size of this text area.
getPreferredSize(int, int)

Determines the preferred size of a text area with the specified number of rows and columns.

getRows()

Gets the number of rows in the text area.

getScrollbarVisibility()

Gets an enumerated value that indicates which scroll bars the text area uses.

insert(String, int)

Inserts the specified text at the specified position in this text area.

paramString()

Returns the parameter string representing the state of this text area.

replaceRange(String, int, int)

Replaces text between the indicated start and end positions with the specified replacement text.

setColumns(int)

Sets the number of columns for this text area.

setRows(int)

Sets the number of rows for this text area.

3.4.4.9.2 Class Name: TextField

Description: A TextField object is a text component that allows for the editing of a single line of text.

Structure:

Specialization_Of: TextComponent

Attributes:

Methods:

addNotify()

Creates the TextField's peer.

echoCharIsSet()

Indicates whether or not this text field has a character set for echoing.

getColumns()

Gets the number of columns in this text field.

getEchoChar()

Gets the character that is to be used for echoing.

getMinimumSize()

Gets the minumum dimensions for this text field.

getMinimumSize(int)

Gets the minumum dimensions for a text field with the specified number of columns.

getPreferredSize()

Gets the preferred size of this text field.

getPreferredSize(int)

Gets the preferred size of this text field with the specified number of columns.

paramString()

Returns the parameter string representing the state of this text field.

processActionEvent(ActionEvent)

Processes action events occurring on this text field by dispatching them to any appropriate objects.

processEvent(AWTEvent)

Processes events on this text field.

setColumns(int)

Sets the number of columns in this text field.

setEchoChar(char)

Sets the echo character for this text field.

3.4.5 Class Name: Cursor

Description: A class to encapsulate the bitmap representation of the mouse cursor.

Structure:

Specialization_Of: Information Presenter

Attributes:

CROSSHAIR_CURSOR
The crosshair cursor type.
DEFAULT_CURSOR
The default cursor type (gets set if no cursor is defined).
E_RESIZE_CURSOR
The east-resize cursor type.
HAND_CURSOR
The hand cursor type.
MOVE_CURSOR
The move cursor type.
N_RESIZE_CURSOR
The north-resize cursor type.
NE_RESIZE_CURSOR
The north-east-resize cursor type.
NW_RESIZE_CURSOR
The north-west-resize cursor type.
predefined
S_RESIZE_CURSOR
The south-resize cursor type.
SE_RESIZE_CURSOR
The south-east-resize cursor type.
SW_RESIZE_CURSOR
The south-west-resize cursor type.
TEXT_CURSOR
The text cursor type.
W_RESIZE_CURSOR
The west-resize cursor type.
WAIT_CURSOR
The wait cursor type.

Methods:

getDefaultCursor()
Return the system default cursor.
getPredefinedCursor(int)
Returns a cursor object with the specified predefined type.
getType()
Returns the type for this cursor.

3.4.6 Class Name: Dimension

Description: The Dimension class encapsulates the width and height of a component in a single object. The class is associated with certain properties of components.

Structure:

Specialization_Of: Information Presenter

Attributes: height
An object's height dimension.
width
An object's width dimension.

Methods: equals(Object)
Checks whether two dimension objects have equal values.
getSize()
Gets the size of this Dimension object.
setSize(Dimension)
Set the size of this Dimension object to the specified size.
setSize(int, int)
Set the size of this Dimension object to the specified width and height.
toString()
Returns a string that represents this Dimension object's values.

3.4.7 Class Name: EventQueue

Description: EventQueue is a platform-independent class that queues events, both from the underlying peer classes and from trusted application classes. There is only one EventQueue for the system.

Structure:
Specialization_Of: Information Presenter

Attributes:

Methods: getNextEvent()
Remove an event from the queue and return it.
peekEvent()
Return the first event without removing it.
peekEvent(int)
postEvent(AWTEvent)
Post a 1.1-style event to the EventQueue.

3.4.8 Class Name: Font

Description: A class that produces font objects.

Structure:
Specialization_Of: Information Presenter

Attributes: BOLD
The bold style constant.
ITALIC
The italicized style constant.
name
The logical name of this font.
PLAIN
The plain style constant.
size
The point size of this font.

style
The style of the font.

Methods: decode(String)
Gets the specified font using the name passed in.
equals(Object)
Compares this object to the specifed object.
getFamily()
Gets the platform specific family name of the font.
getFont(String)
Gets a font from the system properties list.
getFont(String, Font)
Gets the specified font from the system properties list.
getName()
Gets the logical name of the font.
getPeer()
Gets the peer of the font.
getSize()
Gets the point size of the font.
getStyle()
Gets the style of the font.
hashCode()
Returns a hashcode for this font.
isBold()
Indicates whether the font's style is bold.
isItalic()
Indicates whether the font's style is italic.
isPlain()
Indicates whether the font's style is plain.
toString()
Converts this object to a String representation.

3.4.9 Class Name: FontMetrics

Description: A font metrics object, which gives information about the rendering of a particular font on a particular screen.

Structure:

Specialization_Of: Information Presenter

Attributes:

Methods: getAscent()
getDescent()
getLeading()
getMaxAdvance()
charWidth(char ch)
charsWidth(char data[], int off, int len)

3.4.10 Class Name: Graphics

Description: The Graphics class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.

A Graphics object encapsulates state information needed for the basic rendering operations that the GUI supports. This stateinformation includes the following properties:

- The Component object on which to draw.
- A translation origin for rendering and clipping coordinates.
- The current clip.
- The current color.
- The current font.
- The current logical pixel operation function (XOR or Paint).
- The current XOR alternation color (see setXORMode).

Structure:

Specialization_Of: Information Presenter

Attributes:

Methods:

clearRect(int, int, int, int)
: Clears the specified rectangle by filling it with the background color of the current drawing surface.

clipRect(int, int, int, int)
: Intersects the current clip with the specified rectangle.

copyArea(int, int, int, int, int, int)
: Copies an area of the component by a distance specified by dx and dy.

create()
: Creates a new Graphics object that is a copy of this Graphics object.

create(int, int, int, int)
: Creates a new Graphics object based on this Graphics object, but with a new translation and clip area.

dispose()
: Disposes of this graphics context and releases any system resources that it is using.

draw3DRect(int, int, int, int, boolean)
: Draws a 3-D highlighted outline of the specified rectangle.

drawArc(int, int, int, int, int, int)
: Draws the outline of a circular or elliptical arc covering the specified rectangle.

drawBytes(byte[], int, int, int, int)
: Draws the text given by the specified byte array, using this graphics context's current font and color.

drawChars(char[], int, int, int, int)
: Draws the text given by the specified character array, using this graphics context's current font and color.

drawImage(Image, int, int, Color, ImageObserver)

Draws as much of the specified image as is currently available.

drawImage(Image, int, int, ImageObserver)

Draws as much of the specified image as is currently available.

drawImage(Image, int, int, int, int, Color, ImageObserver)

Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.

drawImage(Image, int, int, int, int, ImageObserver)

Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.

drawImage(Image, int, int, int, int, int, int, int, int, Color, ImageObserver)

Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.

drawImage(Image, int, int, int, int, int, int, int, int, ImageObserver)

Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.

drawLine(int, int, int, int)

Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.

drawOval(int, int, int, int)

Draws the outline of an oval.

drawPolygon(int[], int[], int)

Draws a closed polygon defined by arrays of x and y coordinates.

drawPolygon(Polygon)

Draws the outline of a polygon defined by the specified Polygon object.

drawPolyline(int[], int[], int)

Draws a sequence of connected lines defined by arrays of x and y coordinates.

drawRect(int, int, int, int)

Draws the outline of the specified rectangle.

drawRoundRect(int, int, int, int, int, int)

Draws an outlined round-cornered rectangle using this graphics context's current color.

drawString(String, int, int)

Draws the text given by the specified string, using this graphics context's current font and color.

fill3DRect(int, int, int, int, boolean)

Paints a 3-D highlighted rectangle filled with the current color.

fillArc(int, int, int, int, int, int)

Fills a circular or elliptical arc covering the specified rectangle.

fillOval(int, int, int, int)

Fills an oval bounded by the specified rectangle with the current color.

fillPolygon(int[], int[], int)

Fills a closed polygon defined by arrays of x and y coordinates.

fillPolygon(Polygon)

Fills the polygon defined by the specified Polygon object with the graphics context's current color.

fillRect(int, int, int, int)

Fills the specified rectangle.

fillRoundRect(int, int, int, int, int, int)

Fills the specified rounded corner rectangle with the current color.

finalize()

Disposes of this graphics context once it is no longer referenced.

getClip()

Gets the current clipping area.

getClipBounds()

Returns the bounding rectangle of the current clipping area.

getColor()
Gets this graphics context's current color.
getFont()
Gets the current font.
getFontMetrics()
Gets the font metrics of the current font.
getFontMetrics(Font)
Gets the font metrics for the specified font.
setClip(int, int, int, int)
Sets the current clip to the rectangle specified by the given coordinates.
setClip(Shape)
Sets the current clipping area to an arbitrary clip shape.
setColor(Color)
Sets this graphics context's current color to the specified color.
setFont(Font)
Sets this graphics context's font to the specified font.
setPaintMode()
Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color.
setXORMode(Color)
Sets the paint mode of this graphics context to alternate between this graphics context's current color and the new specified color.
toString()
Returns a String object representing this Graphics object's value.
translate(int, int)
Translates the origin of the graphics context to the point (x, y) in the current coordinate system.

3.4.11 Class Name: GUIEvent

Description: The root event class for all GUI events.

Structure:

Specialization_Of: Information Presenter

Attributes:

ACTION_EVENT_MASK
The event mask for selecting action events.
ADJUSTMENT_EVENT_MASK
The event mask for selecting adjustment events.
COMPONENT_EVENT_MASK
The event mask for selecting component events.
consumed
CONTAINER_EVENT_MASK
The event mask for selecting container events.
FOCUS_EVENT_MASK
The event mask for selecting focus events.
id
ITEM_EVENT_MASK
The event mask for selecting item events.
KEY_EVENT_MASK
The event mask for selecting key events.
MOUSE_EVENT_MASK

The event mask for selecting mouse events.

MOUSE_MOTION_EVENT_MASK
The event mask for selecting mouse motion events.

RESERVED_ID_MAX
The maximum value for reserved AWT event IDs.

TEXT_EVENT_MASK
The event mask for selecting text events.

WINDOW_EVENT_MASK
The event mask for selecting window events.

Methods: getID()
Returns the event type.

toString()
Returns a string representation of the object.

3.4.12 Class Name: GUIEventMulticaster

Description: A class which implements efficient and thread-safe multi-cast event dispatching for the GUI. This class will manage an immutable structure consisting of a chain of event listeners and will dispatch events to those listeners. Because the structure is immutable, it is safe to use this API to add/remove listeners during the process of an event dispatch operation.

Structure:

Specialization_Of: Information Presenter

Attributes: a
EventListener a

b
EventListener b

Methods: actionPerformed(ActionEvent)
Handles the actionPerformed event by invoking the actionPerformed methods on listener-a and listener-b.

add(ActionListener, ActionListener)
Adds action-listener-a with action-listener-b and returns the resulting multicast listener.

add(AdjustmentListener, AdjustmentListener)
Adds adjustment-listener-a with adjustment-listener-b and returns the resulting multicast listener.

add(ComponentListener, ComponentListener)
Adds component-listener-a with component-listener-b and returns the resulting multicast listener.

add(ContainerListener, ContainerListener)
Adds container-listener-a with container-listener-b and returns the resulting multicast listener.

add(FocusListener, FocusListener)
Adds focus-listener-a with focus-listener-b and returns the resulting multicast listener.

add(ItemListener, ItemListener)
Adds item-listener-a with item-listener-b and returns the resulting multicast listener.

add(KeyListener, KeyListener)
Adds key-listener-a with key-listener-b and returns the resulting multicast listener.

add(MouseListener, MouseListener)
Adds mouse-listener-a with mouse-listener-b and returns the resulting multicast listener.

add(MouseMotionListener, MouseMotionListener)

Adds mouse-motion-listener-a with mouse-motion-listener-b and returns the resulting multicast listener.

add(TextListener, TextListener)

add(WindowListener, WindowListener)

Adds window-listener-a with window-listener-b and returns the resulting multicast listener.

addInternal(EventListener, EventListener)

Returns the resulting multicast listener from adding listener-a and listener-b together.

adjustmentValueChanged(AdjustmentEvent)

Handles the adjustmentValueChanged event by invoking the adjustmentValueChanged methods on listener-a and listener-b.

componentAdded(ContainerEvent)

Handles the componentAdded container event by invoking the componentAdded methods on listener-a and listener-b.

componentHidden(ComponentEvent)

Handles the componentHidden event by invoking the componentHidden methods on listener-a and listener-b.

componentMoved(ComponentEvent)

Handles the componentMoved event by invoking the componentMoved methods on listener-a and listener-b.

componentRemoved(ContainerEvent)

Handles the componentRemoved container event by invoking the componentRemoved methods on listener-a and listener-b.

componentResized(ComponentEvent)

Handles the componentResized event by invoking the componentResized methods on listener-a and listener-b.

componentShown(ComponentEvent)

Handles the componentShown event by invoking the componentShown methods on listener-a and listener-b.

focusGained(FocusEvent)

Handles the focusGained event by invoking the focusGained methods on listener-a and listener-b.

focusLost(FocusEvent)

Handles the focusLost event by invoking the focusLost methods on listener-a and listener-b.

itemStateChanged(ItemEvent)

Handles the itemStateChanged event by invoking the itemStateChanged methods on listener-a and listener-b.

keyPressed(KeyEvent)

Handles the keyPressed event by invoking the keyPressed methods on listener-a and listener-b.

keyReleased(KeyEvent)

Handles the keyReleased event by invoking the keyReleased methods on listener-a and listener-b.

keyTyped(KeyEvent)

Handles the keyTyped event by invoking the keyTyped methods on listener-a and listener-b.

mouseClicked(MouseEvent)

Handles the mouseClicked event by invoking the mouseClicked methods on listener-a and listener-b.

mouseDragged(MouseEvent)

Handles the mouseDragged event by invoking the mouseDragged methods on listener-a and listener-b.

mouseEntered(MouseEvent)

Handles the mouseEntered event by invoking the mouseEntered methods on listener-a and listener-b.

mouseExited(MouseEvent)

Handles the mouseExited event by invoking the mouseExited methods on listener-a and listener-b.

mouseMoved(MouseEvent)

Handles the mouseMoved event by invoking the mouseMoved methods on listener-a and listener-b.

mousePressed(MouseEvent)

Handles the mousePressed event by invoking the mousePressed methods on listener-a and listener-b.

mouseReleased(MouseEvent)

Handles the mouseReleased event by invoking the mouseReleased methods on listener-a and listener-b.

remove(ActionListener, ActionListener)

Removes the old action-listener from action-listener-l and returns the resulting multicast listener.

remove(AdjustmentListener, AdjustmentListener)

Removes the old adjustment-listener from adjustment-listener-l and returns the resulting multicast listener.

remove(ComponentListener, ComponentListener)

Removes the old component-listener from component-listener-l and returns the resulting multicast listener.

remove(ContainerListener, ContainerListener)

Removes the old container-listener from container-listener-l and returns the resulting multicast listener.

remove(EventListener)

Removes a listener from this multicaster and returns the resulting multicast listener.

remove(FocusListener, FocusListener)

Removes the old focus-listener from focus-listener-l and returns the resulting multicast listener.

remove(ItemListener, ItemListener)

Removes the old item-listener from item-listener-l and returns the resulting multicast listener.

remove(KeyListener, KeyListener)

Removes the old key-listener from key-listener-l and returns the resulting multicast listener.

remove(MouseListener, MouseListener)

Removes the old mouse-listener from mouse-listener-l and returns the resulting multicast listener.

remove(MouseMotionListener, MouseMotionListener)

Removes the old mouse-motion-listener from mouse-motion-listener-l and returns the resulting multicast listener.

remove(TextListener, TextListener)

remove(WindowListener, WindowListener)

Removes the old window-listener from window-listener-l and returns the resulting multicast listener.

removeInternal(EventListener, EventListener)

Returns the resulting multicast listener after removing the old listener from listener-l.

save(ObjectOutputStream, String, EventListener)

saveInternal(ObjectOutputStream, String)

textValueChanged(TextEvent)

windowActivated(WindowEvent)

Handles the windowActivated event by invoking the windowActivated methods on listener-a and listener-b.

windowClosed(WindowEvent)

Handles the windowClosed event by invoking the windowClosed methods on listener-a and listener-b.

windowClosing(WindowEvent)

Handles the windowClosing event by invoking the windowClosing methods on listener-a and listener-b.

windowDeactivated(WindowEvent)

Handles the windowDeactivated event by invoking the windowDeactivated methods on listener-a and listener-b.

windowDeiconified(WindowEvent)

Handles the windowDeiconfied event by invoking the windowDeiconified methods on listener-a and listener-b.

windowIconified(WindowEvent)

Handles the windowIconified event by invoking the windowIconified methods on listener-a and listener-b.

windowOpened(WindowEvent)

Handles the windowOpened event by invoking the windowOpened methods on listener-a and listener-b.

3.4.13 Class Name: Image

Description: The abstract class Image is the superclass of all classes that represent graphical images. The image must be obtained in a platform-specific manner.

Structure:

Specialization_Of: Information Presenter

Attributes:

SCALE_AREA_AVERAGING

Use the Area Averaging image scaling algorithm.

SCALE_DEFAULT

Use the default image-scaling algorithm.

SCALE_FAST

Choose an image-scaling algorithm that gives higher priority to scaling speed than smoothness of the scaled image.

SCALE_REPLICATE

Use the image scaling algorithm embodied in the ReplicateScaleFilter class.

SCALE_SMOOTH

Choose an image-scaling algorithm that gives higher priority to image smoothness than scaling speed.

UndefinedProperty

The UndefinedProperty object should be returned whenever a property which was not defined for a particular image is fetched.

Methods:

flush()

Flushes all resources being used by this Image object.

getGraphics()

Creates a graphics context for drawing to an off-screen image.

getHeight(ImageObserver)

Determines the height of the image.

getProperty(String, ImageObserver)

Gets a property of this image by name.

getScaledInstance(int, int, int)
Creates a scaled version of this image.
getSource()
Gets the object that produces the pixels for the image.
getWidth(ImageObserver)
Determines the width of the image.

3.4.14 Class Name: Insets

Description: An Insets object is a representation of the borders of a container. It specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.

Structure:
Specialization_Of: Information Presenter

Attributes: bottom
The inset from the bottom.
left
The inset from the left.
right
The inset from the right.
top
The inset from the top.

Methods: clone()
Create a copy of this object.
equals(Object)
Checks whether two insets objects are equal.
toString()
Returns a String object representing this Insets object's values.

3.4.15 Class Name: LayoutManager

Description: Defines the LayoutManager class that is used to layout components in Containers.

Structure:
Specialization_Of: Information Presenter

Attributes:

Methods: addLayoutComponent(String, Component)
Adds the specified component with the specified name to the layout.
layoutContainer(Container)
Lays out the container in the specified panel.
minimumLayoutSize(Container)
Calculates the minimum size dimensions for the specified panel given the components in the specified parent container.
preferredLayoutSize(Container)
Calculates the preferred size dimensions for the specified panel given the components in the specified parent container.
removeLayoutComponent(Component)
Removes the specified component from the layout.

3.4.16 Class Name: MediaTracker

Description: The MediaTracker class is a utility class to track the status of a number of media objects. Media objects could include audio clips as well as images.

Structure:

Specialization_Of: Information Presenter

Attributes:

ABORTED
: Flag indicating that the downloading of some media was aborted.

COMPLETE
: Flag indicating that the downloading of media was completed successfully.

ERRORED
: Flag indicating that the downloading of some media encountered an error.

LOADING
: Flag indicating some media is currently being loaded.

Methods:

addImage(Image, int)
: Adds an image to the list of images being tracked by this media tracker.

addImage(Image, int, int, int)
: Adds a scaled image to the list of images being tracked by this media tracker.

checkAll()
: Checks to see if all images being tracked by this media tracker have finished loading.

checkAll(boolean)
: Checks to see if all images being tracked by this media tracker have finished loading.

checkID(int)
: Checks to see if all images tracked by this media tracker that are tagged with the specified identifier have finished loading.

checkID(int, boolean)
: Checks to see if all images tracked by this media tracker that are tagged with the specified identifier have finished loading.

getErrorsAny()
: Returns a list of all media that have encountered an error.

getErrorsID(int)
: Returns a list of media with the specified ID that have encountered an error.

isErrorAny()
: Checks the error status of all of the images.

isErrorID(int)
: Checks the error status of all of the images tracked by this media tracker with the specified identifier.

removeImage(Image)
: Remove the specified image from this media tracker.

removeImage(Image, int)
: Remove the specified image from the specified tracking ID of this media tracker.

removeImage(Image, int, int, int)
: Remove the specified image with the specified width, height, and ID from this media tracker.

statusAll(boolean)
: Calculates and returns the bitwise inclusive OR of the status of all media that are tracked by this media tracker.

statusID(int, boolean)
Calculates and returns the bitwise inclusive OR of the status of all media with the specified identifier that are tracked by this media tracker.
waitForAll()
Starts loading all images tracked by this media tracker.
waitForAll(long)
Starts loading all images tracked by this media tracker.
waitForID(int)
Starts loading all images tracked by this media tracker with the specified identifier.
waitForID(int, long)
Starts loading all images tracked by this media tracker with the specified identifier.

3.4.17 Class Name: MenuComponent

Description: The abstract class MenuComponent is the superclass of all menu-related components. In this respect, the class MenuComponent is analogous to the abstract superclass Component for GUI components.

Structure:
Specialization_Of: Information Presenter

Attributes:

Methods: dispatchEvent(AWTEvent)
getFont()
Gets the font used for this menu component.
getName()
Gets the name of the menu component.
getParent()
Returns the parent container for this menu component.
paramString()
Returns the parameter string representing the state of this menu component.
processEvent(AWTEvent)
Processes events occurring on this menu component.
removeNotify()
Removes the menu component's peer.
setFont(Font)
Sets the font to be used for this menu component to the specified font.
setName(String)
Sets the name of the component to the specified string.
toString()
Returns a representation of this menu component as a string.

3.4.17.1 Class Name: MenuBar

Description: The MenuBar class encapsulates the platform's concept of a menu bar bound to a frame.

Structure:
Specialization_Of: MenuComponent

Attributes:

Methods: add(Menu)
Adds the specified menu to the menu bar.

addNotify()
Creates the menu bar's peer.
deleteShortcut(MenuShortcut)
Deletes the specified menu shortcut.
getHelpMenu()
Gets the help menu on the menu bar.
getMenu(int)
Gets the specified menu.
getMenuCount()
Gets the number of menus on the menu bar.
getShortcutMenuItem(MenuShortcut)
Gets the instance of MenuItem associated with the specified MenuShortcut object, or null if none has been specified.
remove(int)
Removes the menu located at the specified index from this menu bar.
remove(MenuComponent)
Removes the specified menu component from this menu bar.
removeNotify()
Removes the menu bar's peer.
setHelpMenu(Menu)
Sets the help menu on this menu bar to be the specified menu.
shortcuts()
Gets an enumeration of all menu shortcuts this menu bar is managing.

3.4.17.2 Class Name: MenuItem

Description: All items in a menu must belong to the class MenuItem The default MenuItem object embodies a simple labeled menu item.

Structure:

Specialization_Of: MenuComponent

Attributes:

Methods:

addNotify()
Creates the menu item's peer.
deleteShortcut()
Delete any MenuShortcut object associated with this menu item.
disableEvents(long)
Disables event delivery to this menu item for events defined by the specified event mask parameter.
enableEvents(long)
Enables event delivery to this menu item for events to be defined by the specified event mask parameter.
getActionCommand()
Gets the command name of the action event that is fired by this menu item.
getLabel()
Gets the label for this menu item.
getShortcut()
Get the MenuShortcut object associated with this menu item,
isEnabled()
Checks whether this menu item is enabled.
paramString()

Returns the parameter string representing the state of this menu item.

processActionEvent(ActionEvent)

Processes action events occurring on this menu item, by dispatching them to any appropriate objects.

processEvent(AWTEvent)

Processes events on this menu item.

setActionCommand(String)

Sets the command name of the action event that is fired by this menu item.

setEnabled(boolean)

Sets whether or not this menu item can be chosen.

setLabel(String)

Sets the label for this menu item to the specified label.

setShortcut(MenuShortcut)

Set the MenuShortcut object associated with this menu item.

3.4.17.2.1 Class Name: CheckboxMenuItem

Description: This class represents a check box that can be included in a menu. Clicking on the check box in the menu changes its state from "on" to "off" or from "off" to "on."

Structure:

Specialization_Of: MenuItem

Attributes:

Methods:

addNotify()

Creates the peer of the checkbox item.

getSelectedObjects()

Returns the an array (length 1) containing the checkbox menu item label or null if the checkbox is not selected.

getState()

Determines whether the state of this check box menu item is "on" or "off."

paramString()

Returns the parameter string representing the state of this check box menu item.

processEvent(AWTEvent)

Processes events on this check box menu item.

processItemEvent(ItemEvent)

Processes item events occurring on this check box menu item by dispatching them to any appropriate objects.

setState(boolean)

Sets this check box menu item to the specifed state.

3.4.17.2.2 Class Name: Menu

Description: A Menu object is a pull-down menu component that is deployed from a menu bar. A menu can optionally be a tear-off menu. A tear-off menu can be opened and dragged away from its parent menu bar or menu. It remains on the screen after the mouse button has been released. The mechanism for tearing off a menu is platform dependent, since the look and feel of the tear-off menu is determined by its peer. On platforms that do not support tear-off menus, the tear-off property is ignored.

Each item in a menu must belong to the MenuItem class.

Structure:

Specialization_Of: MenuItem

Attributes:

Methods: add(MenuItem)
Adds the specified menu item to this menu.
add(String)
Adds an item with the specified label to this menu.
addNotify()
Creates the menu's peer.
addSeparator()
Adds a separator line, or a hypen, to the menu at the current position.
getItem(int)
Gets the item located at the specified index of this menu.
getItemCount()
Get the number of items in this menu.
insert(MenuItem, int)
Inserts a menu item into this menu at the specified position.
insert(String, int)
Inserts a menu item with the specified label into this menu at the specified position.
insertSeparator(int)
Inserts a separator at the specified position.
isTearOff()
Indicates whether this menu is a tear-off menu.
paramString()
Gets the parameter string representing the state of this menu.
remove(int)
Removes the menu item at the specified index from this menu.
remove(MenuComponent)
Removes the specified menu item from this menu.
removeAll()
Removes all items from this menu.
removeNotify()
Removes the menu's peer.

3.4.17.2.2.1 Class Name: PopupMenu

Description: A class that implements a menu which can be dynamically popped up at a specified position within a component.

Structure:

Specialization_Of: Menu

Attributes:

Methods: addNotify()
Creates the popup menu's peer.
show(Component, int, int)
Shows the popup menu at the x, y position relative to an origin component.

3.4.18 Class Name: MenuShortcut

Description: A class which represents a keyboard accelerator for a MenuItem.

Structure:

Specialization_Of: Information Presenter

Attributes:

Methods: equals(MenuShortcut)

Returns whether this MenuShortcut is the same as another: equality is defined to mean that both MenuShortcuts use the same key and both either use or don't use the SHIFT key.

getKey()

Return the raw keycode of this MenuShortcut.

paramString()

toString()

Returns an internationalized description of the MenuShortcut.

usesShiftModifier()

Return whether this MenuShortcut must be invoked using the SHIFT key.

3.4.19 Class Name: Point

Description: The Point class represents a location in a two-dimensional (x, y) coordinate space.

Structure:

Specialization_Of: Information Presenter

Attributes: x

The x coordinate.

y

The y coordinate.

Methods: equals(Object)

Determines whether two points are equal.

getLocation()

Returns the location of this point.

hashCode()

Returns the hashcode for this point.

move(int, int)

Moves this point to the specificed location in the (x, y) coordinate plane.

setLocation(int, int)

Changes the point to have the specificed location.

setLocation(Point)

Sets the location of the point to the specificed location.

toString()

Returns a representation of this point and its location in the (x, y) coordinate space as a string.

translate(int, int)

Translates this point, at location (x, y), by dx along the x axis and dy along the y axis so that it now represents the point (x + dx, y + dy).

3.4.20 Class Name: Polygon

Description: The Polygon class encapsulates a description of a closed, two-dimensional region within a coordinate space. This region is bounded by an arbitrary number of line segments, each of which is one side of the polygon. Internally, a polygon comprises of a list of (x, y) coordinate pairs, where each pair defines a vertex of the polygon, and two successive pairs are the endpoints of a line that is a side of the polygon. The first and final pairs of (x, y) points are joined by a line segment that closes the polygon.

Structure:

Specialization_Of: Information Presenter

Attributes: bounds

npoints
: The total number of points.

xpoints
: The array of x coordinates.

ypoints
: The array of y coordinates.

Methods: addPoint(int, int)
: Appends a point to this polygon.

contains(int, int)
: Determines whether the specified point is contained by this polygon.

contains(Point)
: Determines whether the specified point is inside the Polygon.

getBounds()
: Gets the bounding box of this polygon.

translate(int, int)
: Translates the vertices by deltaX along the x axis and by deltaY along the y axis.

3.4.21 Class Name: PrintJob

Description: An abstract class which initiates and executes a print job. It provides access to a print graphics object which renders to an appropriate print device.

Structure:

Specialization_Of: Information Presenter

Attributes:

Methods: end()
: Ends the print job and does any necessary cleanup.

finalize()
: Ends this print job once it is no longer referenced.

getGraphics()
: Gets a Graphics object that will draw to the next page.

getPageDimension()
: Returns the dimensions of the page in pixels.

getPageResolution()
: Returns the resolution of the page in pixels per inch.

lastPageFirst()
: Returns true if the last page will be printed first.

3.4.22 Class Name: Rectangle

Description: A rectangle specifies an area in a coordinate space that is defined by the rectangle's top-left point (x, y) in the coordinate space, its width, and its height.

Structure:

Specialization_Of: Information Presenter

Attributes: height

The height of the rectangle.

width

The width of the rectangle.

x

The x coordinate of the rectangle.

y

The y coordinate of the rectangle

Methods: add(int, int)

Adds a point, specified by the integer arguments newx and newy, to a rectangle.

add(Point)

Adds a point to this rectangle.

add(Rectangle)

Adds a rectangle to this rectangle.

contains(int, int)

Checks whether this rectangle contains the point at the specified location (x, y).

contains(Point)

Checks whether this rectangle contains the specified point.

equals(Object)

Checks whether two rectangles are equal.

getBounds()

Gets the bounding rectangle of this rectangle.

getLocation()

Returns the location of this rectangle.

getSize()

Gets the size (width and height) of this rectangle.

grow(int, int)

Grows the rectangle both horizontally and vertically.

hashCode()

Returns the hashcode for this rectangle.

intersection(Rectangle)

Computes the intersection of this rectangle with the specified rectangle.

intersects(Rectangle)

Determines whether this rectangle and the specified rectangle intersect.

isEmpty()

Determines whether this rectangle is empty.

setBounds(int, int, int, int)

Sets the bounding rectangle of this rectangle to the specified values for x, y, width, and height.

setBounds(Rectangle)

Sets the bounding rectangle of this rectangle to match the specified rectangle.

setLocation(int, int)

Moves the rectangle to the specified location.

setLocation(Point)

Moves the rectangle to the specified location.

setSize(Dimension)

Sets the size of this rectangle to match the specified dimension.

setSize(int, int)

Sets the size of this rectangle to the specified width and height.

toString()

Returns a string representation of this rectangle and its values.

translate(int, int)

Translates the rectangle the indicated distance, to the right along the x coordinate axis, and downward along the y coordinate axis.

union(Rectangle)

Computes the union of this rectangle with the specified rectangle.

3.4.23 Class Name: Toolkit

Description: This class is the abstract superclass of all actual implementations of the GUI Toolkit. Subclasses of Toolkit are used to bind the various components to particular native toolkit implementations.

The methods defined by Toolkit are the "glue" that joins the platform-independent classes in the GUI with their counterparts in native GUI. Some methods defined by Toolkit query the native operating system directly.

Structure:

Specialization_Of: Information Presenter

Attributes:

Methods: beep()

Emits an audio beep.

checkImage(Image, int, int, ImageObserver)

Indicates the construction status of a specified image that is being prepared for display.

createButton(Button)

Creates this toolkit's implementation of Button using the specified peer interface.

createCanvas(Canvas)

Creates this toolkit's implementation of Canvas using the specified peer interface.

createCheckbox(Checkbox)

Creates this toolkit's implementation of Checkbox using the specified peer interface.

createCheckboxMenuItem(CheckboxMenuItem)

Creates this toolkit's implementation of CheckboxMenuItem using the specified peer interface.

createChoice(Choice)

Creates this toolkit's implementation of Choice using the specified peer interface.

createComponent(Component)

Creates a peer for a component or container.

createDialog(Dialog)

Creates this toolkit's implementation of Dialog using the specified peer interface.

createFileDialog(FileDialog)

Creates this toolkit's implementation of FileDialog using the specified peer interface.

createFrame(Frame)

Creates this toolkit's implementation of Frame using the specified peer interface.

createImage(byte[])

Creates an image which decodes the image stored in the specified byte array.

createImage(byte[], int, int)

Creates an image which decodes the image stored in the specified byte array, and at the specified offset and length.

createImage(ImageProducer)
Creates an image with the specified image producer.
createLabel(Label)
Creates this toolkit's implementation of Label using the specified peer interface.
createList(List)
Creates this toolkit's implementation of List using the specified peer interface.
createMenu(Menu)
Creates this toolkit's implementation of Menu using the specified peer interface.
createMenuBar(MenuBar)
Creates this toolkit's implementation of MenuBar using the specified peer interface.
createMenuItem(MenuItem)
Creates this toolkit's implementation of MenuItem using the specified peer interface.
createPanel(Panel)
Creates this toolkit's implementation of Panel using the specified peer interface.
createPopupMenu(PopupMenu)
Creates this toolkit's implementation of PopupMenu using the specified peer interface.
createScrollbar(Scrollbar)
Creates this toolkit's implementation of Scrollbar using the specified peer interface.
createScrollPane(ScrollPane)
Creates this toolkit's implementation of ScrollPane using the specified peer interface.
createTextArea(TextArea)
Creates this toolkit's implementation of TextArea using the specified peer interface.
createTextField(TextField)
Creates this toolkit's implementation of TextField using the specified peer interface.
createWindow(Window)
Creates this toolkit's implementation of Window using the specified peer interface.
getColorModel()
Determines the color model of this toolkit's screen.
getDefaultToolkit()
Gets the default toolkit.
getFontList()
Returns the names of the available fonts in this toolkit.
getFontMetrics(Font)
Gets the screen metrics of the font.
getFontPeer(String, int)
Creates this toolkit's implementation of Font using the specified peer interface.
getImage(String)
Returns an image which gets pixel data from the specified file.
getImage(URL)
Returns an image which gets pixel data from the specified URL.
getMenuShortcutKeyMask()
Determines which modifier key is the appropriate accelerator key for menu shortcuts.
getNativeContainer(Component)
Give native peers the ability to query the native container given a native component.
getPrintJob(Frame, String, Properties)
Gets a PrintJob object which is the result of initiating a print operation on the toolkit's platform.
getProperty(String, String)
Gets a property with the specified key and default.
getScreenResolution()
Returns the screen resolution in dots-per-inch.
getScreenSize()
Gets the size of the screen.
getSystemClipboard()

Gets an instance of the system clipboard which interfaces with clipboard facilities provided by the native platform.

getSystemEventQueue()

Get the application's or applet's EventQueue instance.

getSystemEventQueueImpl()

loadSystemColors(int[])

Fills in the integer array that is supplied as an argument with the current system color values.

prepareImage(Image, int, int, ImageObserver)

Prepares an image for rendering.

sync()

Synchronizes this toolkit's graphics state.

3.4.24 Class Name: Video

Description: The abstract class Video is the superclass of all classes that represent video streams. The video must be obtained in a platform-specific manner.

Structure:

Specialization_Of: Information Presenter

Attributes: SCALE_AREA_AVERAGING

Use the Area Averaging video scaling algorithm.

SCALE_DEFAULT

Use the default video -scaling algorithm.

SCALE_FAST

Choose an video -scaling algorithm that gives higher priority to scaling speed than smoothness of the scaled video.

SCALE_REPLICATE

Use the video scaling algorithm embodied in the ReplicateScaleFilter class.

SCALE_SMOOTH

Choose an video -scaling algorithm that gives higher priority to video smoothness than scaling speed.

UndefinedProperty

The UndefinedProperty object should be returned whenever a property which was not defined for a particular video is fetched.

Methods: flush()

Flushes all resources being used by this video object.

getGraphics()

Creates a graphics context for drawing to an off-screen video.

getHeight(ImageObserver)

Determines the height of the video.

getProperty(String, ImageObserver)

Gets a property of this video by name.

getScaledInstance(int, int, int)

Creates a scaled version of this video.

getSource()

Gets the object that produces the pixels for the video.

getWidth(ImageObserver)

Determines the width of the video.

3.5 Example Application of the Information Presenter Class Hierarchy

3.5.1 Introduction

This example presents a high-level application of the IPC hierarchy. Specifically, one instance of the General Range Intelligent Display System (GRIDS) is mapped to the IPC. The mapping presented is not unique or exhaustive. The flexibility of the IPC permits and supports many variants of such mappings.

3.5.2 Basic Assumptions

GRIDS defines a display screen as the whole screen area (i.e., viewable area) of a graphics CRT. The display screen is logically broken into two areas: The standard viewport and the graphics page (Figure 25).

The standard viewport is a system defined area of the display screen that displays general use information about the system and the functional capabilities that can be executed. The standard viewport is logically subdivided into the standard status structure display and the standard menu structure. The standard menu structure displays the selectable and executable functions that can be used to modify or change the displays seen on the graphics page (Figure 26).

The graphics page is a user defined area of the display screen that displays the real-time data. The graphics page is logically divided into graphics structures, which are logically divided into graphics components (Figure 27).

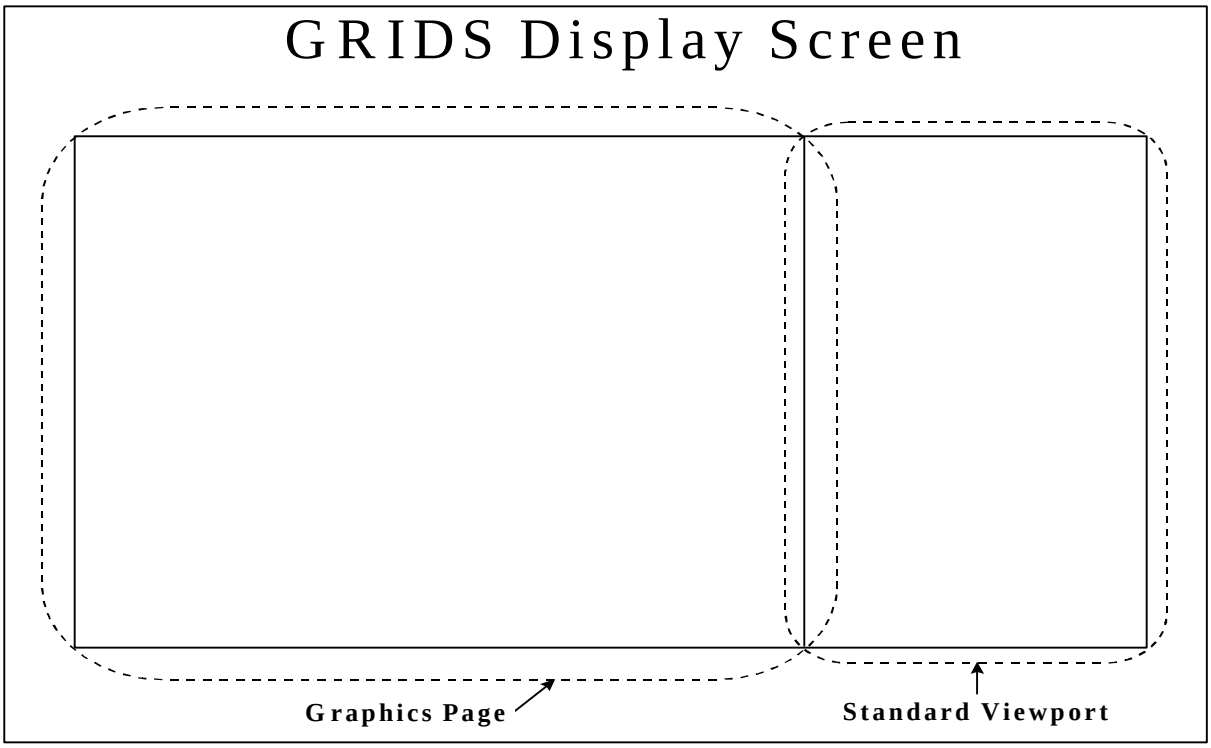

Figure 25. Grids Display Screen (1)

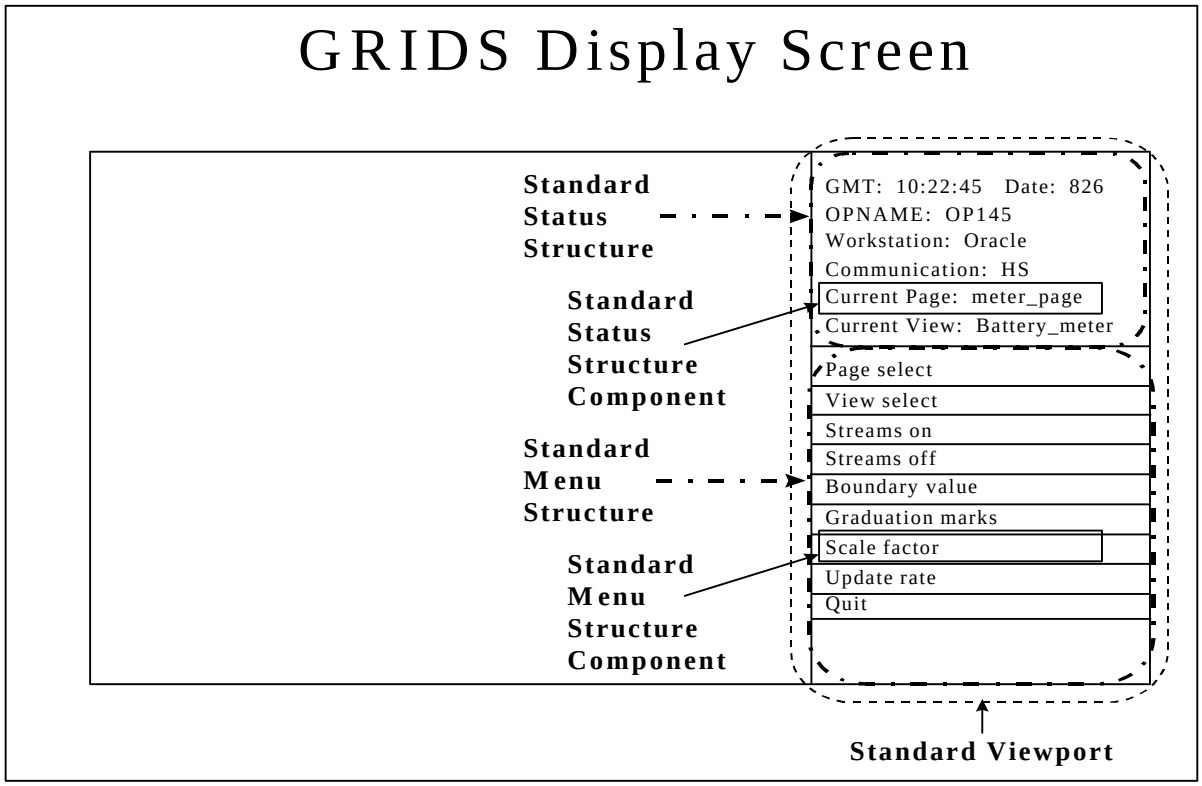

Figure 26. GRIDS Display Screen (2)

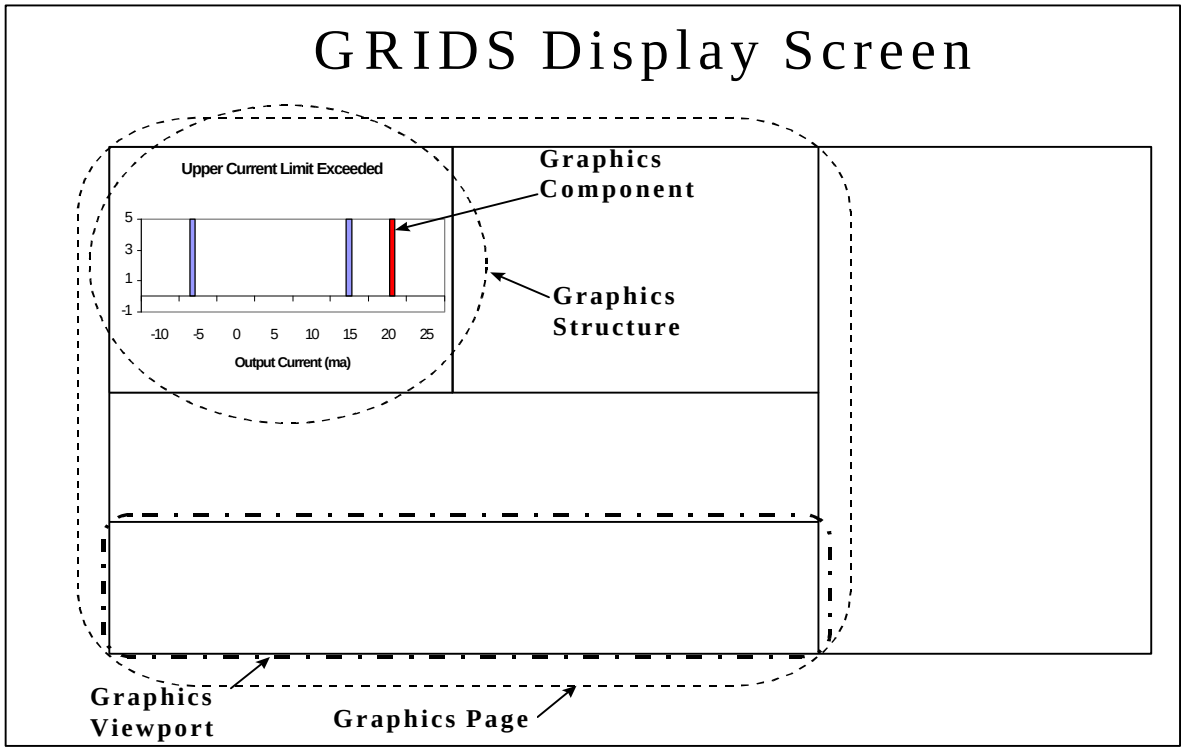

Figure 27. GRIDS Display Screen (3)

3.5.3 Example

The example GRIDS display to be mapped onto the IPC is shown in Figure 28. The display contains many of the GRIDS components identified in Figures 25-27. The mapping will be constrained by the IPC sub-hierarchy (or specialization) given in Figure 29. Finally, Figures 25-27 relate specific GRIDS components to corresponding IPC components.

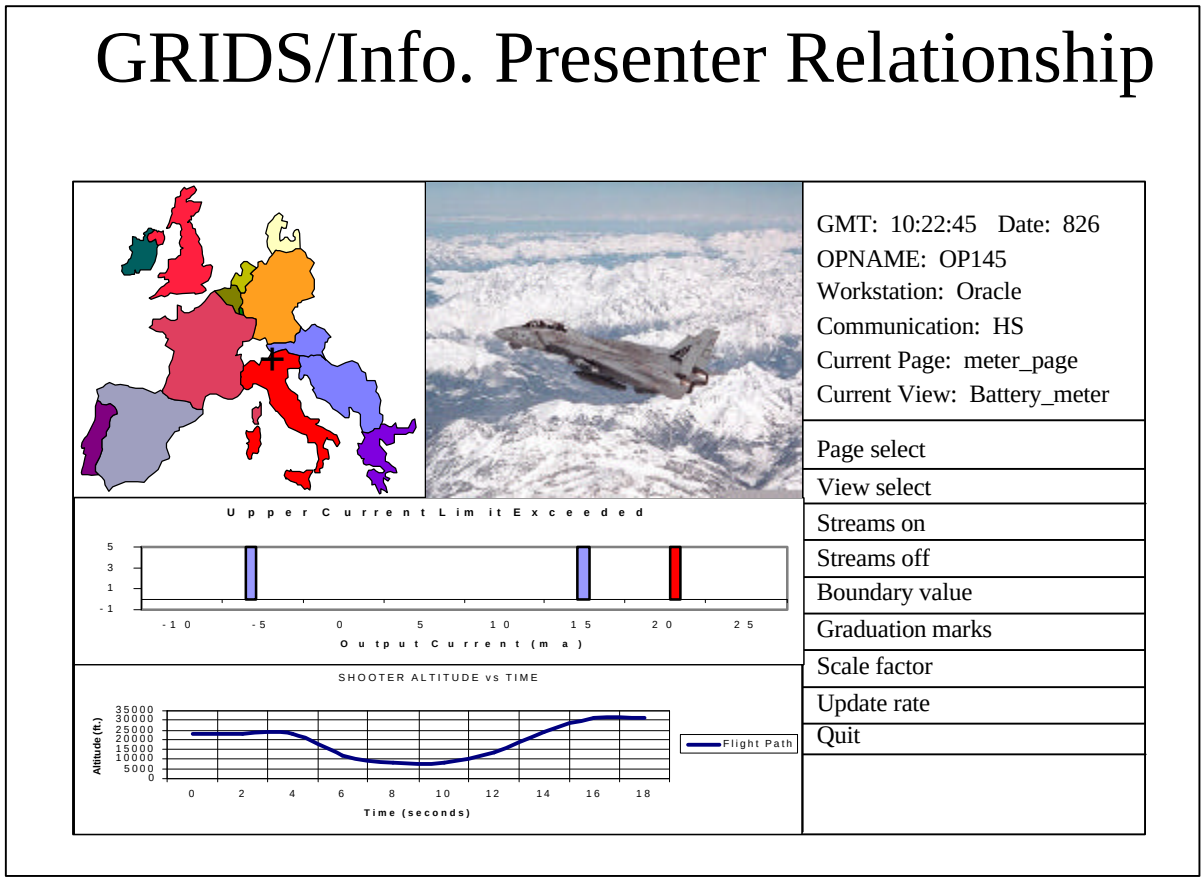

Figure 28. GRIDS/Info. Presenter Relationship (1)

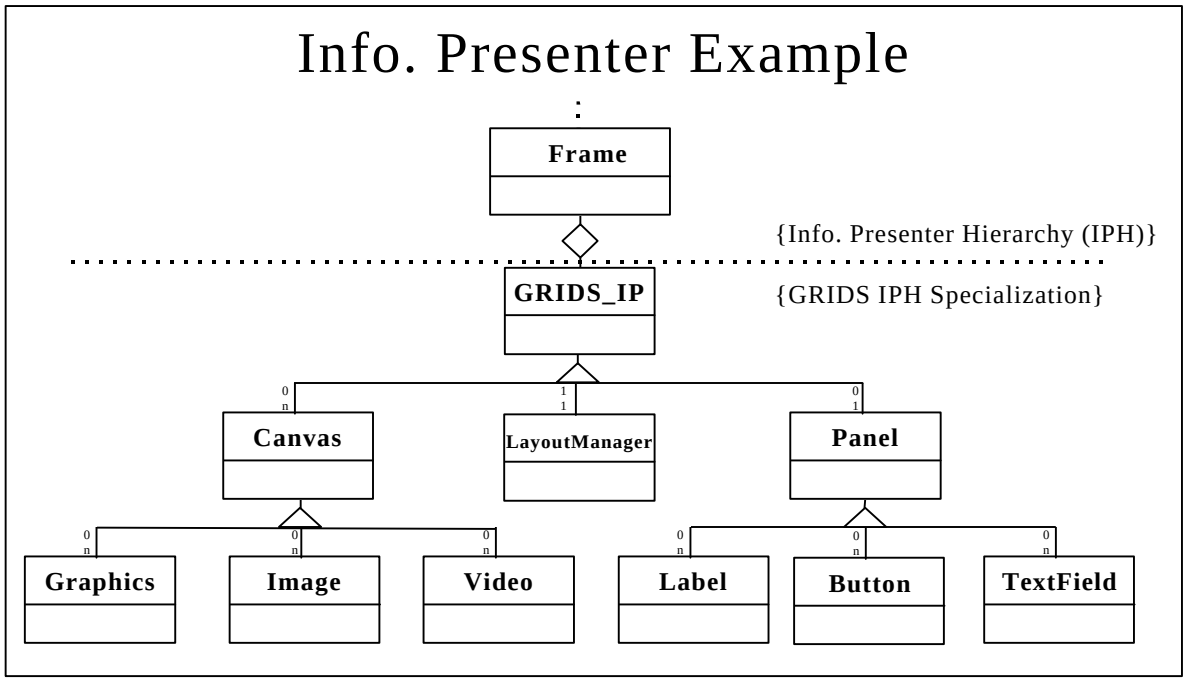

Figure 29. Info. Presenter Example

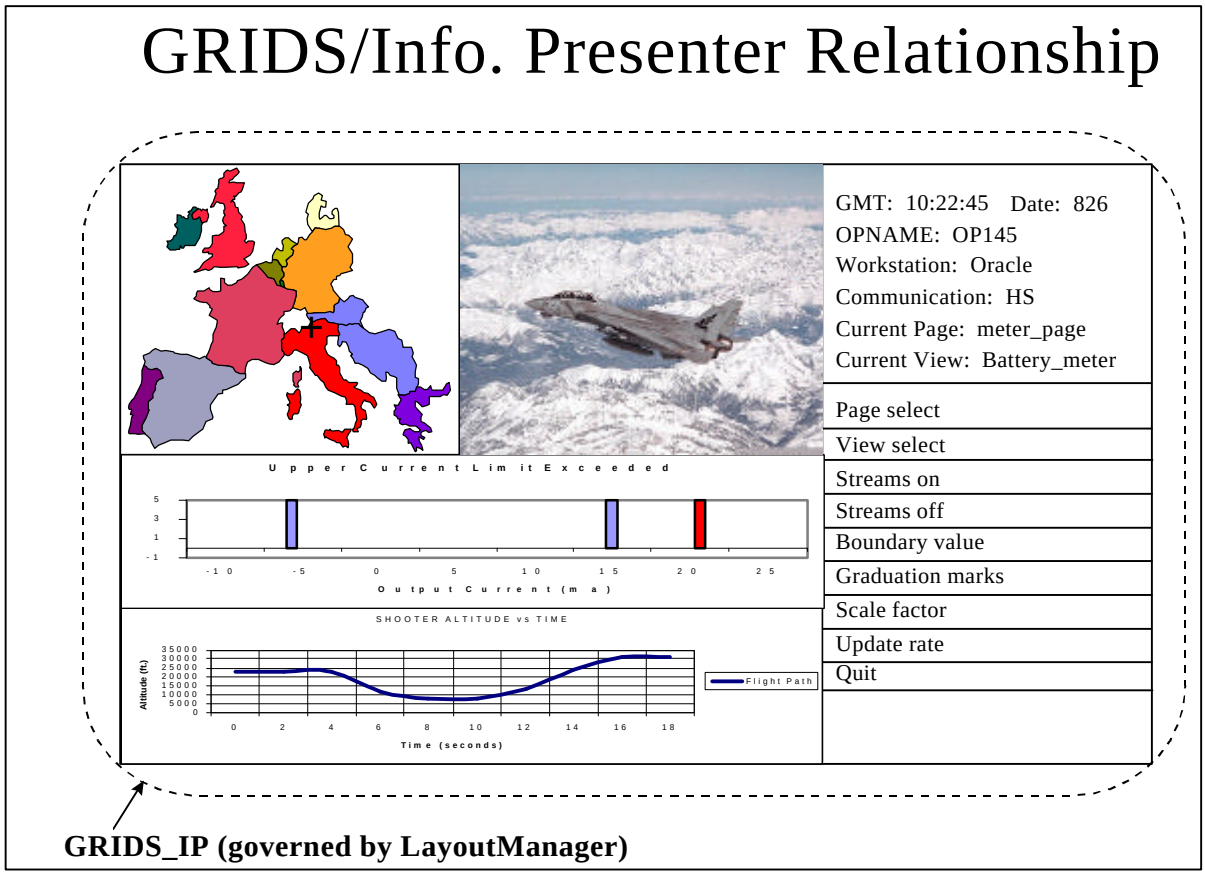

Figure 30. GRIDS/Info. Presenter Relationship (1)

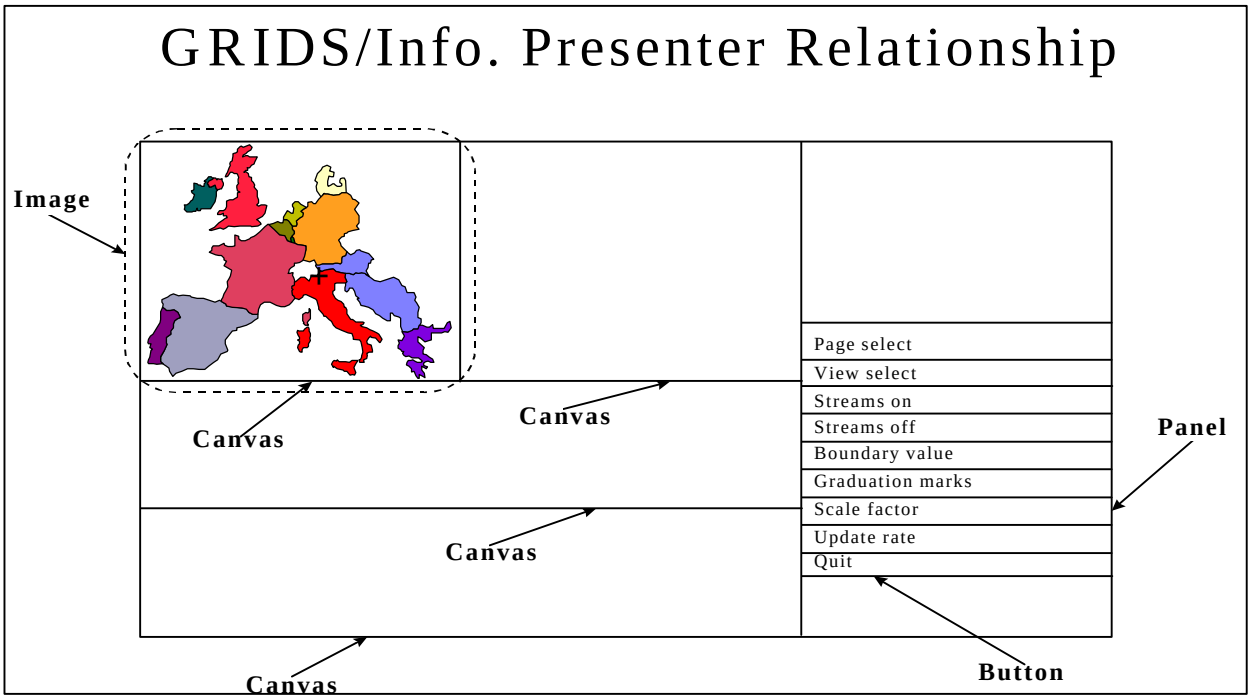

Figure 31. GRIDS/Info. Presenter Relationship (2)

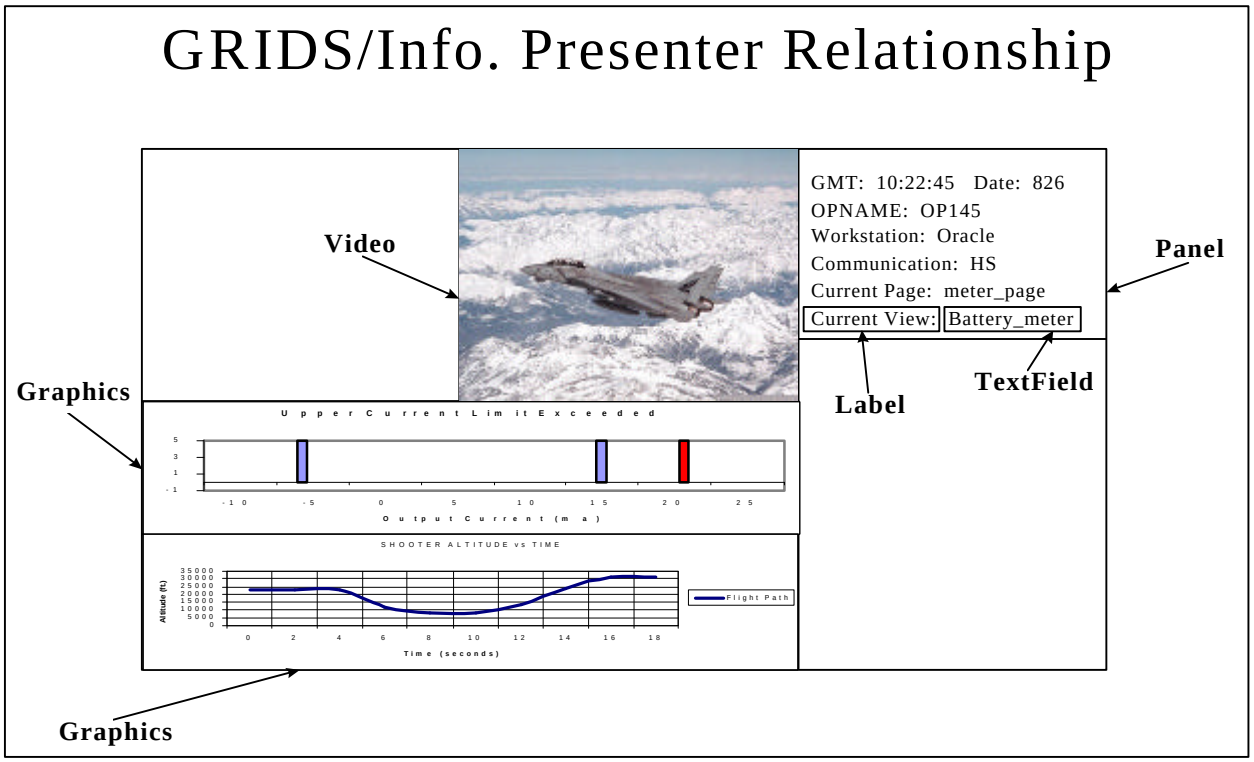

Figure 32. GRIDS/Info. Presenter Relationship (3)

4.0 TENA Core

4.1 Introduction

At the most abstract level, the invariant part of every instance of the Logical Range can be considered to consist of the system infrastructure services and mandatory system applications. This invariant base is called the TENA Core. It is represented graphically in Figure 33. One of the most important aspects of the TENA Core is that it is standard across all instances of the Logical Range, with one minor exception: some portions of the standard mandatory applications may be customized to suit particular range or facility circumstances when necessary. This is the only portion of the TENA Core that is considered "site-specific" with the remainder being entirely invariant. This site-specific portion of the otherwise invariant mandatory applications will be fully specified prior to implementation so that a clear distinction is made between the TENA Core invariant sections and the sections that can be customized.

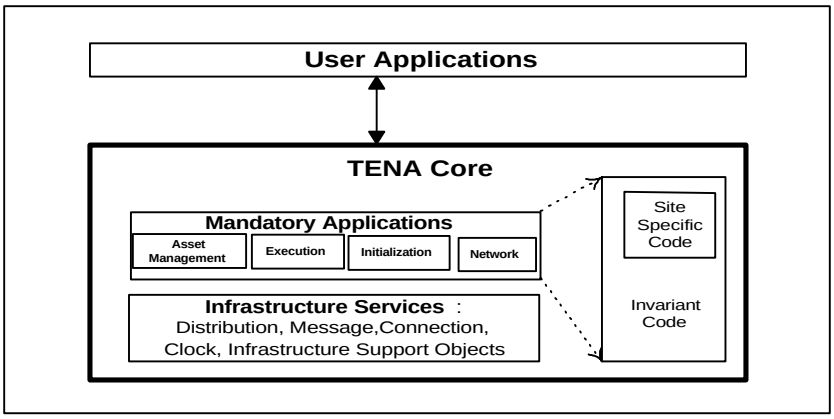

Figure 33. TENA Core

Customization that accommodates site uniqueness will not interfere with or alter the standardization of the interfaces or functioning of these mandatory applications. It will enable sites to tailor applications for unique site configurations (hardware-specific), procedures (site-specific operational requirements), and practices (site-specific logs and record-keeping).

4.2 Purpose of the TENA Core

The purpose of the TENA Core is to provide a means for exercise participants to communicate with each other and to provide mechanisms for the management of the coordinated operation of an instance of the logical range. All exercise processes and business aspects of facility operations are integrated through capabilities provided by the TENA Core. The TENA Core provides the means for logical range components to interoperate while maintaining a level of isolation among them, limiting

knowledge of the details of where other components reside, how to communicate with them, and the details of their implementation.

The TENA Core manages the overall system operation and integrates the various elements available in the TENA object model. It provides services that are general in nature and distributed across the entire system.

Figure 9 is a conceptual model of the TENA Core. A more detailed model would show that there are numerous instances of the TENA Core distributed about the facilities. Each TENA Core is a collection of distributed services that coordinate their activities as required. This coordination is effected via the normal data communications mechanisms that the TENA Core provides to any service requester. The diagram shown below depicts a single instance of the TENA Core showing some of the internal structures and interfaces.

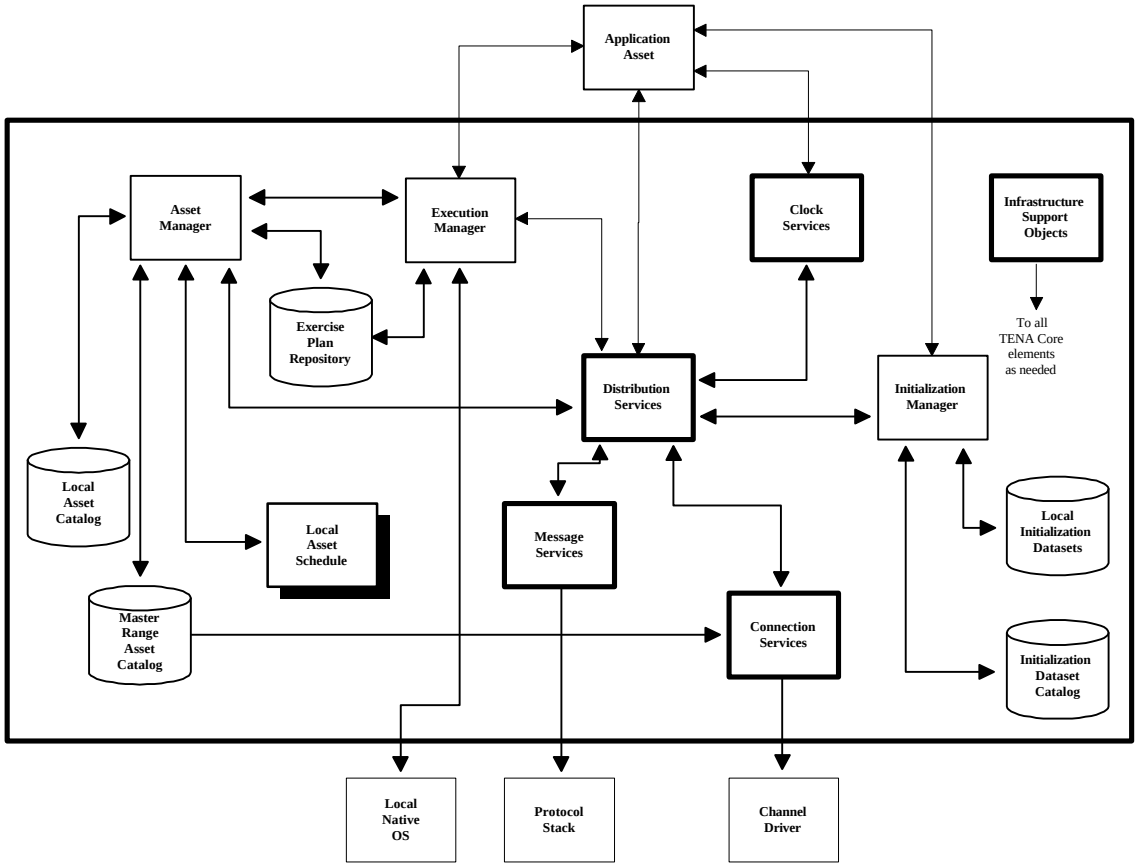

Figure 34. Conceptual Model of the TENA Core

4.3 Sample Facilities Functional Partitions

Figure 35 is a comprehensive but abstract view that depicts sample functional partitions identified for the facility system. It situates these partitions within the system and provides a perspective on how the facility system is related to other modeling and simulation systems (M&S) as well as to systems foreign to either the facility or M&S domains. A special type of surrogate, the TENA Bridge, illustrates the integration of legacy facility systems with the TENA Core. This diagram does not imply a single user interface or Human Computer Interface (HCI) application. Applications within functional partitions

are provided with user interfaces as appropriate. These interfaces are conceptually part of the HCI partition.

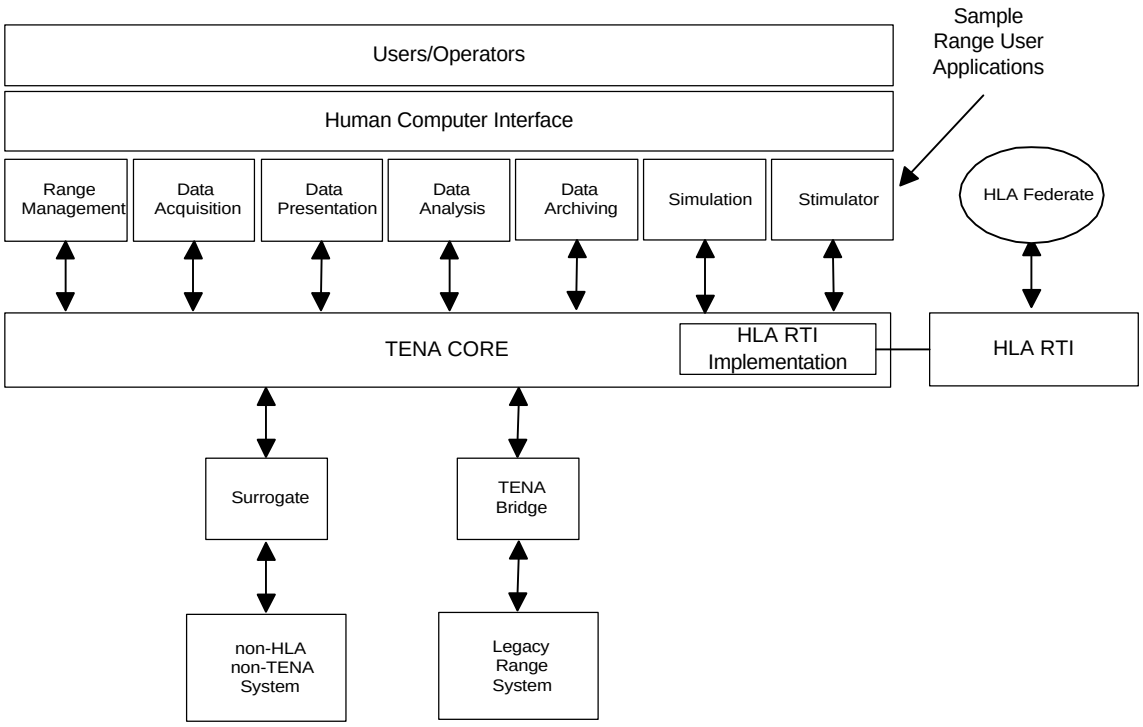

Figure 35. Sample Functional Partitions

The identification of partitions within the applications area, that is, above the TENA Core, as shown, is somewhat notional. This section of the document focuses on the Core of the system and defers consideration of most of the functional details of the applications. The TENA Core is the basic *glue* that integrates the system. It is intended to provide a set of powerful and open capabilities that applications can use to coordinate their activities. A great deal of flexibility facilitates rapid reconfiguration of asset sets for test conduct. It anticipates the need to change configurations, representations, and test specific properties on short notice before and during test execution. It also responds to the need to support integration of various application tools used before, during, and after a test to satisfy business processes used by facilities. Later discussions will elaborate on these and provide definition of what mechanisms, policies, and standards the architecture provides to manage these capabilities.

The top-level view shown in Figure 35 consists of the following partitions:

Users/Operators:

Consists of the personnel who operate and use the facilities. Covers all phases from exercise planning through after exercise analysis and reporting as well as O&M personnel. (see Object Model Personnel class)

Human Computer Interface**:**

Consists of libraries and frameworks used to constitute parts of applications which communicate with users and operators. Concerned primarily with display technologies and associated processing, e.g. GUIs. (see Object Model Information Presenter class)

Facility Management:

Consists of several applications which are used to define and plan exercises, configure logical facilities, manage inventories, perform billing and cost recovery, generate usage reports, monitor maintenance, manage schedules, produce cost estimates, search and match facility requirements to facility capabilities and assets, etc. Most of the business process activities and assets would be contained within this partition. Those areas would include exercise and test management and control, as well as the facility management applications indicated.(see Object Model Logical Range Planning Tool class and others)

Data Acquisition:

Includes all aspects of sensing and preliminary processing of data on the facilities. Includes all sensors and telemetry systems and devices. (see Object Model Sensor class)

Data Analysis:

Applications used to perform various analyses on datasets collected on the facilities. (see Object Model Analyzer class)

Data Presentation:

Performs processing and provides devices used for the display of data to users during any phase of an exercise and after completion of the exercise. Provides capabilities to debrief exercise participants, conduct after-action reviews, and generate reports for customer use. These applications and equipment handle all of the processing behind the scenes of the HCI. (see Object Model Information Presenter class)

Data Archiving:

Applications handle all storage and retrieval of datasets produced by or in association with facility exercises or otherwise used by facility assets. (see Object Model , various class methods)

Simulations:

Modeling applications used to provide synthetic battlespace elements for interaction with other facility assets. (see Object Model Primary Resource class)

TENA Core:

Fundamental component of architecture used to provide system wide or common services and mandatory applications that integrate the systems and assets on facilities into a cooperative enterprise for the purpose of satisfying particular intended purposes. The TENA Core encapsulates all operating systems, utilities, communications protocols, communications and coordination facilities, etc. which provide the environment in which applications execute and coordinate their activities.

HLA Run Time Infrastructure:

Those elements of the HLA RTI needed for interaction with federations of modeling and simulation systems. The relationship of TENA to the HLA is discussed in more detail below.

Surrogate:

A stand in for systems which are neither TENA compliant nor HLA compliant but with which facility assets must interact. The surrogate is an interface between different architectures.

TENA Bridge:

Legacy systems that comprise current facility assets would probably all integrate with the Facility System via a TENA Bridge. This wrapper technology can be implemented in a number of ways. This might range from a stand-alone bridge unit to a set of translation/interface services integrated with legacy code. Various considerations including costs to modify existing systems will drive the choice. It is anticipated that at the start of the transition to the TENA range facilities would primarily consist of existing assets that are bridged to a copy of the TENA core. The TENA core instances will connect with each other via appropriate networks to integrate dispersed legacy resources.

4.4 TENA and the HLA

One of the goals of the TENA is to allow Test and Evaluation and Training facilities to interoperate with M&S facilities. The primary means of achieving this is through the High Level Architecture. The first approach taken here was to develop the facility system as a federate, from the perspective of the HLA. The facility system then federates with M&S systems supporting the relevant federation object models. However, on closer analysis it is apparent that some of the functionality that is required for the TENA Software Infrastructure, specifically that involved in moving data of interest between participants, could be satisfied by an HLA run time infrastructure implementation.

A suitable RTI implementation can be embedded within the TENA Core to allow interoperation with M&S systems either directly, when the RTI implementations are compatible, or through a bridge federate, when the RTI implementations are not compatible, different time management strategies are in use, or security considerations require it. The details of this are significant at the implementation level and are not discussed further in this document.

The Information Class Catalog contains all of the information about the data exchanged by participants either with themselves or with other federates in an HLA federation. In the case where the facility system is bridged to a federation then an approach is to describe that facility system as a federate with a Simulation Object Model describing its capabilities. The Simulation Object Model which characterizes the facility system federate can be extracted directly from this catalog. If the facility system is viewed as a federation itself then the Federation Object Model for this federation may also be extracted directly from the Information Class Catalog as well as the Simulation Object Models for the participating federates.

4.5 Description of theTENA Core

4.5.1 Introduction

The TENA Core is composed of Information Management Services and mandatory applications that are standard across all instances of the TENA facilities. The Information Management Services are:

- Distribution Services
- Message Services
- Connection Services
- Clock Services
- Infrastructure Support Objects

The required or mandatory applications are:

- Network Manager
- Asset Manager
- Execution Manager
- Initialization Manager

The first three service groups, Distribution, Message, and Connection Services, are responsible for the object based subscription service which provides for information movement within the system, both data and control. Distribution Services provides the only procedural interface within the infrastructure services which applications have access to. Access to all other assets including all other services of the infrastructure is via this object based subscription service.

The remaining two groups, Clock Services and Infrastructure Support Objects provide capabilities for coordinating time on the logical range and internal functioning of the infrastructure.

The four required applications, the Asset Manager, the Execution Manager, the Initialization Manager, and the Network Manager, are responsible for supporting the planning, scheduling, and execution of

tests/exercises on instances of the logical range, managing any required initialization data, and managing communications resources.

It should be noted that there is a collection of tradeoffs (advantages, disadvantages) that were considered when determining whether a required capability of TENA should be defined as an infrastructure service or as a required application that complements and completes the services. This current partitioning does not mean that some of the required applications could not be incorporated as a service of the infrastructure, only that it does not seem mandatory. Several of the capabilities currently defined as applications were originally envisioned as infrastructure services, and may return to that category as more detailed information becomes available through future verification, validation, prototyping, and testing of TENA.

4.5.2 Information Management Services

4.5.2.1 System Information Model

This section provides a high level view of the basis for how the system coordinates activities. It refers to various TENA Core services and applications that are described in more detail in the following sections. The information model used to describe all information movement within the system, whether data or control, is an object based subscription service. All information exchanged within the system is described in terms of classes and their attributes. Sources of data advertise the availability of this data by issuing a Publish service call that establishes the ability to produce information as described. The Publish service requires the publisher to identify the class of information and the particular attributes defined for the class that it is offering. The attributes announced must be defined in the Information Class Catalog and can include any attribute or set of attributes defined in the catalog for the indicated class but does not have to include all attributes listed for that class.

Users who require information that is described in the Information Class Catalog announce this desire to the infrastructure by executing a Subscribe service call. The subscriber describes the class and the particular attributes of that class which are desired. The infrastructure uses the information from the Publish and Subscribe service invocations to manage the distribution of information within the system.

4.5.2.1.1 Information Classes

As indicated, the information class is the basis for describing data exchanged within the system. Except for the interface between Distribution Services and other system elements all of the interfaces captured in class definitions are implemented via the subscription service with Distribution Services orchestrating the physical movement of the data. Interfaces that are depicted in this document are usually logical interfaces that describe the information exchanged between system elements, the exception being interfaces with Distribution Services. In most cases, for the sake of clarity the involvement of Distribution Services in the exchange and its procedural interfaces are omitted from the interface diagrams. The Information Class Catalog (ICC) can be thought of as the information model of a system. All of the information that is contained in the various system interfaces is described within this catalog along with indications of how the data is intended to be used and what data is required to be produced by assets. It also identifies information, mostly events, to which assets are required to subscribe. In this latter case the asset is required to respond to the information in a meaningful way.

This is primarily intended for event classes, described below, used to send advisories, requests, or commands to system components.

The Information Class Catalog (ICC) can be thought of as the information model of the system. All of the information that is exchanged by assets is described within this catalog. Information is included with the class definitions that describes how the class and its attributes is intended to be used along with constraints on its use. Many of the classes defined describe system events, which are sporadic communications, that assets are required to subscribe to. These are used to carry control information, i.e. commands, as well as requests and advisories.

The service descriptions provided in Appendix C of this document list the details of the interface parameters, exceptions, etc. The ICC provides, in essence, the implementation level details for all of these interfaces except those of Distribution Services. The implementation details for Distribution Services interfaces, which are procedural interfaces, are provided in the Application Program Interface (API) in Appendix D. The ICC contains descriptions of classes that are defined by infrastructure designers as well as by application designers. It is meant to be a living database that contains information used directly by implementations during run-time as well as information of interest to users indicating how the classes are to be employed and some restrictions on their use.

The classes defined in the Information Class Catalog fall into one of two categories depending on their use. They may be considered as static classes, that is, classes for which attribute values are, in general, provided repeatedly and where the expectation is that the underlying object that the information models or is persistent. For this use of an information class the publisher announces the intent to provide data for specific class instances by invoking the RegisterInstance service for each class instance. The parameter returned by this service is a system unique instance identifier that is used to unambiguously reference the class instance throughout the system. Attribute values are provided by the publisher via the Update service. An update can include values for any attribute that has been announced as being published but need not include values for all attributes. Essentially attributes are independent in terms of the time or condition of update of their values except where specific agreements have been reached about the bundling of attribute values (described in the ICC). Unpublishing the class requires that any instances registered first be removed. If the Unpublish service is invoked while class instances are still registered an exception will be thrown and the publisher will be required to explicitly remove these instances.

The second category of use of a class is as an event. For this type of class the publisher provides attribute values by using the SendEvent service. This is assumed to be a transient instantiation of the class that lasts until the event has been delivered to subscribers. The publisher does not register an instance for the class in this case and attribute values are entrained in the event delivery rather than provided via an update. Attributes can only be included if they have been identified as being published by the publisher of the event. This class is intended to be used to provide a means to define and use transient phenomena that occur at unpredictable times. Many of the control mechanisms, e.g., commands, in the system are implemented using this method.

Components that require a particular class of information make this known to the infrastructure via the Subscribe service. The subscriber provides the class identifier and a list of the class attributes in which it is interested. The infrastructure uses this information to manage the distribution of data within the

system. The instance identifier of the instance is provided as a parameter of this service so that the subscribers can identify the attribute values that are provided for the instance in an update. A RemoveInstance service is provided to delete registered instances when they become irrelevant or fall out of scope.

4.5.2.1.2 Information Channels

The establishment of an interested pair, a publisher and a subscriber, defines an information channel in the system. Each publisher-subscriber pair defines a unique virtual channel whose contents are determined by the subscriber's attribute list. There is no *a priori* limit on the number of information channels that may exist for any class, and each of the channels will contain at least one attribute. The information channel is created when the matching of the publisher and subscriber first occurs and the channel persists until one of the pair divests its interest in the information exchange. These channels are virtual information links; they describe a shared interest but must be mapped to a physical channel for information to be moved.

The direct interface between applications assets and the infrastructure is between the asset and Distribution Services. This is a procedural interface with the asset directly invoking methods exported by Distribution Services and Distribution Services directly calling functions the asset has provided callback addresses for. The system procedural interfaces are shown in Figure 36.

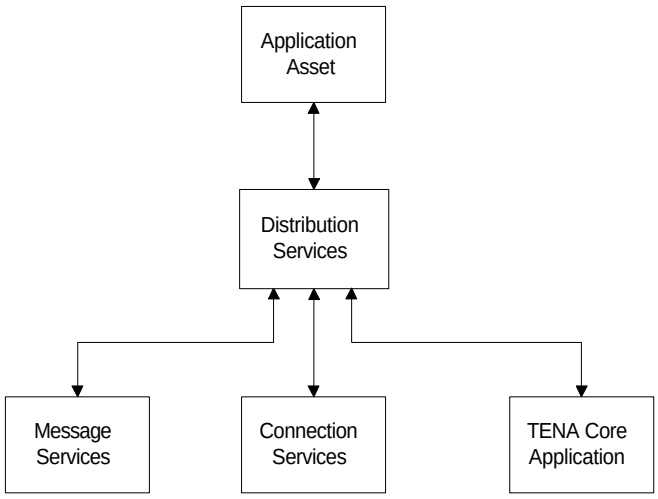

Figure 36 System Procedural Interfaces

All other application asset to infrastructure interfaces are message based interfaces with no direct procedural connection. Distribution Services assumes responsibility for distributing this information. Interfaces between TENA Core components that are local to the same infrastructure as well as

interfaces between components of separate infrastructure instances use Distribution Services to manage the data transfer, as for other system information transfers. These interfaces, including those which involve infrastructure services other than Distribution Services, are mechanized via information classes. Infrastructure services captured in information classes consist primarily of event classes and are managed like transactions.

4.5.2.1.3 Transactions

A number of the interactions between elements of the system, particularly those that involve TENA Core elements can be described as transactions. A transaction consists of one or more events that are coupled together to provide a complete and atomic service. The carriers of these transactions are events, or instances of event classes, which are dispatched and received in response to higher level functional requirements. Many of the services provided by the infrastructure will involve requesters with infrastructure elements as actors engaging in a transaction. For example, many services involve a request or command followed by some response. For purposes of the TENA these interactions are considered transactions. As described in this document, the events are dispatched using the SendEvent service provided by Distribution Services. Most of the services provided by the TENA Core are realized by defining one or more event classes with attributes, described as parameters in the service descriptions, and placing them in the Information Class Catalog.

The implementation of TENA Core components will provide a means of ensuring that transactions either complete, in which case state variables are modified in consequence, or they fail, in which case there are no permanent changes to state variables made. Sometimes the commit can be inferred from the completion of actions and sometimes an actual commit event is required. The ICC describes any coupling of events into transactions and requirements for committing a transaction.

4.5.2.1.4 Mapping to Communications Channels

For information to be transmitted within the system it is necessary for each information channel to be mapped to a communications channel. A communications channel is a physical conduit for movement of data. It involves specific hardware (including circuits, gateways, network interfaces, etc.) and, usually, software that transmits data from one location to another. A particular communications channel may involve the creation of logical channels that share communications resources.

The publisher of a class indicates the level of communications support it requires by providing quality of service parameters that the infrastructure uses to determine the actual communications channel required. The information channels established for the class published are then mapped to the communications channel for delivery. These parameters are used to convey communications requirements from general purpose, shared channels through special purpose, dedicated channels. Publishers and subscribers of information determine the specific communications channel used (when known *a priori*) for delivery of the information, or specify the channel characteristics required. Subscribers and publishers will need to coordinate their requirements and ensure that the Information Class Catalog reflects the agreements on delivery of data. In cases where essentially the same information needs to be delivered to multiple subscribers where it is not possible to use the same

quality of service on each information channel, publishers will define multiple classes that can be published with different quality of service that satisfy the subscriber needs.

System architectures and implementations of TENA Core components may have multiple ways of representing quality of service. At its simplest, a type designation could be assigned to each known channel or channel type with publishers selecting which type they prefer. More sophisticated approaches can express quality of service in terms of parameters, such as minimum bandwidth, maximum latency, mean latency, error rate, reliability, etc. The TENA Enterprise must, however, agree on a common set of representations that are supported throughout the enterprise. This is necessary to enable components to be reused across system and facility boundaries. Differences in type referents may reflect how tightly constrained a component is by its communications requirements. For example a specification for a particular channel may indicate that only that channel is sufficient for its needs whereas a specification for bandwidth, latency, reliability, etc. could be satisfied by a number of different channels. A particular implementation or instance of the infrastructure may not be able to recognize or act on all quality of service parameters. All implementations will be designed to respond to unrecognized and unsupported parameters by issuing exceptions and providing default parameters for communications channels involved. Infrastructure instances may not have access to communications assets supporting a particular quality or mix of qualities. They will respond by throwing an exception and resorting to default parameters for the effected communications channels.

A requested quality of service may not always be available. Users may have to decide whether a compromise between desired and available service is possible. Infrastructure implementations may choose to attempt a best match between available resources and the requested quality but will probably always rely, to some extent, on a user's decision about the suitability of a communications channel for a particular use.

4.5.2.1.5 Physical Channels and Circuits.

Physical channels are implemented using communications circuits. These range from simple cables connecting two points to facilities provided by common carriers. Where multiplexing is supported multiple physical channels may be mapped to an individual circuit.

The fundamental physical building block for interconnecting assets is the circuit. This includes the physical medium plus protocols for signaling, access control, error management, etc. These circuits are divided into two use categories based on their visibility in the system. Private circuits are those that directly connect two assets. There is no infrastructure component located in or attached to the circuit and there is no requirement for using infrastructure services to move data through the channel. Consequently, any functionality that is provided by infrastructure components or the underlying platform is not available. This means that any conventions for moving information in the circuit or requirements for formatting of data are the responsibility of the assets that are connected to the circuit. The test plan, or a suitable management application, indicates which assets are to be connected, the identity of the circuit or required circuit characteristics, and how the collection is connected. Allocation and configuration of circuits is the responsibility of the Network Manager. Connection information, obtained from the Network Manager, is provided to the assets so that they know how to access the low level hardware and driver software to use the circuit.

Public circuits are circuits that can be used by one or more pairs of communicating assets. These circuits are connected through Distribution Services and make use of either Message Services or Connection Services for low level management of circuit access. Users issue Publish or Subscribe service invocations to announce to Distribution Services their need for a particular kind of circuit via the quality of service parameters. Information channels are mapped to the circuits based upon the various publications and subscriptions and information is moved through the circuits via infrastructure services. These public circuits may be either dedicated or shared.

Dedicated public circuits are intended to be used for special application needs and are managed via Connection Services. The circuits are defined either in the test plan or via user direction through an appropriate application asset. The Network Manager asset is responsible for creating or allocating these circuits, configuring them as directed, and connecting them to indicated hardware ports. The connection information required to access the channels is provided by the Network Manager. Assets indicate via quality of service parameters which protocol sets, if any, are to be utilized to encapsulate access to the circuits. At its most primitive, channels may be accessed directly through interface hardware.

Shared public circuits are based on using the communications capabilities provided by the underlying platform using well known network technologies and protocol sets. Low level management of these circuits is via Message Services and they are accessible to any asset on an as needed basis. There are no restrictions on the number of users, pattern of use, or loading beyond that imposed by circuit capacity and system requirements. Distribution Services is used for managing all communications to these circuits from applications and infrastructure components. These circuits are normally set up prior to powering on platforms, are permanently configured into the system, and are available after the platforms operating system is booted up. They are directly supported by the operating system kernel.

4.5.2.1.6 Filtering

For a variety of reasons a specific subscriber to a class may need to control the information delivered to it. The means to do this are provided by filters installed within an information channel. In general, subscribers activate one or more filters as required to limit data flowing within the channel. They do this by using the ActivateFilter service and provide filter parameters that identify the type of filter and the value for the specific parameters required by the filter type. There is no inherent limitation on the number or type of filters that may be defined or used within any channel. However, some filter types may be meaningless for a particular information channel.

It is expected that filters will be used to control the amount of data flowing within a channel to reduce resources required by components and manage loading of communications circuits. They may also be used to match components operating at different cyclic rates, interface synchronous and asynchronous components, and simplify the design of components by providing commonly used data filtering techniques within the infrastructure.

There are no limitations on when filters may be activated and deactivated. The value of filter parameters can also be modified to change the filter characteristics at any time. The presence of filters and the value of parameters have no effect on the distribution of information about the existence of class instances.

Filters that are established within information channels perform no transformations on information flowing through the channel. They merely gate the information depending on whether the passing criteria are met or not. Users will decide if filtering is appropriate in an information channel. Some data streams cannot be altered in any way, even by simple traffic control. The gating criteria are identified by the parameters supplied during the filter installation. Based on the type of filter selected, certain information class attributes are identified as being mapped to dimensions of the filter space. Values for the minimal and maximal values of the dimension variable that define the mapping of the filter gate to the particular dimension are also provided. The dimensions provided define a space of 1 or more dimensions.

In general, the information that passes through the channel need not have any connection to the attributes mapped to filter space dimensions. In determining the passage of information the filter operates on the latest values of the dimensional variables that are available to it. If the dimensional variables are mapped from the attributes of an instance generated by the publisher then the latest values are available immediately from the local source. If the attributes are generated by an instance not locally available the infrastructure instance that executes the filter will subscribe for the required class and attributes and obtain the latest values through the normal update service.

4.5.2.1.7 Mandatory and Default Publishing and Subscribing

The current concept for communications between TENA Core elements located in different infrastructure instances, between applications assets, and between applications assets and TENA Core elements is to route those communications through Distribution Services. As a consequence of this and for additional reasons there is a body of information classes that will be defined and used for publication and subscription purposes by all components that are capable of accessing an instance of the infrastructure. Each type of component will have to publish and subscribe some subset of these classes as a default set of information class interests.

These classes will carry the requests and responses for infrastructure services as well as required management information to be used for facility management and performance monitoring. Other classes will provide events that can be used for control of assets. Facility management applications and infrastructure services will make use of these events as appropriate to control the operation of the system's assets. All assets that are capable will publish information on their status for use in determining the state of assets. All of the mandatory publishing and subscribing requirements are described in the Information Class Catalog.

Facilities will, from time to time, define assets that will be used for test purposes to diagnose problems with facility assets or assess their operational readiness. These test assets will publish and subscribe information as defined by maintainers to provide information flow for stimulating test instruments.

4.5.2.2 Distribution Services:

4.5.2.2.1 Description

Manages the distribution of data between facility assets and infrastructure components. This service makes use of message services and connection services as appropriate to distribute the data between assets, and uses event classes to distribute information between infrastructure components. This component provides a subscription oriented service for data based on classes and class attributes. Data transfers are supported with a variety of Quality of Service factors (QOS) on both shared and dedicated channels. Components instantiate the object classes they publish as required by their operation. At some appropriate time they register these instances with the TENA core by a service call to distribution services. Distribution services then advises all subscribers to the class of the existence of the instances through the discover service. This is one of the services that invoke a callback in a system asset.

4.5.2.2.2 Assumptions

This system relies on an object-based concept. Distribution of data throughout the system is aligned with this model of the system. System components establish classes that define the objects or instances of the classes from which the system is constructed. In terms of data transfer, distribution services expects that components requiring service provide definitions of classes of objects and classes of events. These class definitions contain attributes that are the carriers of the state information that is distributed within the system.

Instance of event classes (events) are created as required and dispatched into the system. There is no registration or discovery associated with these class instances. They are considered discrete, self contained, and non-persistent.

The minimum granularity to which distribution is managed is the attribute level. Components announce their intent to produce data for general consumption by using the publish service. The class and attributes are identified to the infrastructure as part of the service. Additional parameters are provided to define conditions of delivery and certain data characteristics. Consumers of data identify their needs by using the subscribe service, identifying the class and attributes desired and requirements for rate of delivery, etc.

It should be noted that the Quality of Service requested by the publisher of data must be prearranged with all planned subscribers since no mechanism is supported for modifying a particular QOS during operations. Different QOS required by different subscribers will be supported by separate classes with separate QOS being published. A failure of a link satisfying a QOS during an exercise may be circumvented by subscribing to a backup QOS with a backup link.

Components are under no obligation to publish all of the information they contain. They announce availability via a publish service for only those classes and attributes they wish to make available outside the component. They can further specify access limitations on the data via parameters of the service. These access limitations establish subsets of components that can access a particular class. Distribution services enforces the access restrictions by checking a subscriber provided password against a publisher provided password. This password is arrived at outside of the Infrastructure so it can not be requested during an exercise. It must be established as a part of test scheduling when the all participants of an exercise are involved. The password protected data is limited on a class basis.

4.5.2.2.3 Rationale

Standardization of communication within the system is critical. A new data or communication interface is accomplished by defining a Class and Attributes, then Publishing and Subscribing to them, with no new software required.

All Infrastructure components and all facility assets will communicate through Distribution Services.

The intent of this concept is that Distribution Services is used by all system components and for interfaces within the system. All facility assets that are not infrastructure assets accomplish all their communications through distribution services with rare exception. Infrastructure components use distribution services for all communications that travel between infrastructure instances. Because of this reliance of all other services on Distribution Service, it must be the first infrastructure service activated. It must also be self-initializing (i.e., no communication with Initialization Manager which is not yet active).

4.5.2.2.4 Services Provided

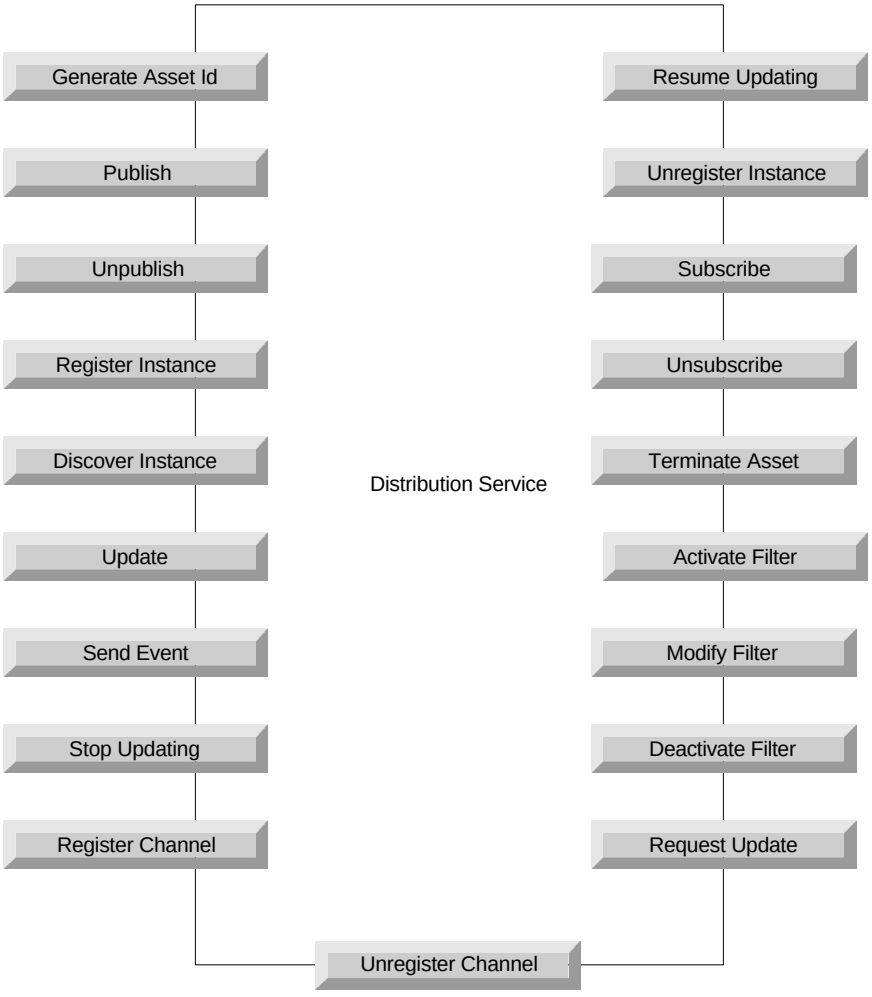

Figure37. Distribution Service - Services Provided

4.5.2.2.4.1 Generate Asset ID

This service is used to generate a system unique identifier for an asset. This would be used prior to adding an asset to the master asset catalog.

4.5.2.2.4.2 Publish

A service used to announce the intention or capability of providing information described by parameters for use by system components. This is in effect a declaration that this particular class of information is going to be available to all other components of the logical range. Those logical range participants interested in receiving this information must subscribe to it. This is true of both static classes (persistent and changing data) and event classes (non-persistent or discrete).

4.5.2.2.4.3 Unpublish

Used by a system component to announce its intent to cease providing the information for the selected class described by the parameters. All instances of the Class which were registered must first be removed before executing this service.

4.5.2.2.4.4 Register Instance

This service is used by a system component to announce the existence of an instance of some class that it publishes. This establishes the intent to provide state data about this instance. A publish declares an intention to provide data of a certain class. This service indicates that a particular instance of that class is now available. Those components that have subscribed to this class will now begin receiving data for this instance of that class.

4.5.2.2.4.5 Discover Instance

This service is an infrastructure initiated service that is invoked for all subscribers to a class when an instance of the class has been registered. This informs subscribers that a class of data they have subscribed to now has data available for a particular instance of that class and they will now begin to receive that data.

4.5.2.2.4.6 Update

Used by publishers of information to provide values for the indicated attributes of the indicated Class instance. Update services propagates the values provided to all components that have expressed a desire to receive the information provided that the discovery criteria for the receiver has been met and the information passes through any filters established in the delivery channel. [When a Class is published, Distribution Service provides a Delivery Handle to the publisher to be associated with the future Updates of the Class instance. Although the Delivery Handle is not passed as a parameter for the Update service, it is used by the updating asset in directing or delivering the update in accordance with the syntax of the programming language being used. The API for Distribution Service will contain specific information on the use of the Delivery Handle. The use of the Delivery Handle permits

Distribution Services to set up entries to lower levels in the stack of infrastructure services to allow lower latency updates over dedicated public channels through Connection Services.]

4.5.2.2.4.7 Send Event

Used by a component to dispatch a discrete event into the system. The event will be propagated to all indicated components that have expressed a desire to receive the event. When an event class has been published and subscribed to, this service distributes the event to all subscribers.

4.5.2.2.4.8 Stop Updating

This service is used to advise the publisher to cease updating a class that it is publishing. This occurs when there are no subscribers to a particular class of data.

4.5.2.2.4.9 Resume Updating

This service is used to notify the publisher of a class to resume updating. This indicates that there is at least one subscriber to the class.

4.5.2.2.4.10 Unregister Instance

This service is invoked to announce that an owner of a class instance will cease to provide information about that instance. It is propagated to subscribers to a class to advise them that an instance of the class has been removed from the system. This indicates that no more information about this instance will be delivered.

4.5.2.2.4.11 Subscribe

A service used by any component that wishes to announce a need for information described in the parameters. Subscribers are notified of the registration of all class instances that meet the discovery criteria specified in this service.

4.5.2.2.4.12 Unsubscribe

Used by system components to remove themselves from the distribution lists for the specified information.

4.5.2.2.4.13 Activate Filter

The use of Publish and Subscribe services creates information channels within the system. Each pairing of a publisher with a subscriber defines one of these logical channels. For many reasons related to performance of communications networks, resource contention at devices with service network connections, cost of network bandwidth, etc. there is a need to limit information transfer.

The establishing of interest groups via Publish and Subscribe is one form of limitation of transfer since the services establish a minimal connection network among components. However, many cases require additional filtering on information channels.

This service is used to activate a filter in an information channel to effect a desired degree of limitation. It is the subscriber to data who activates a filter to protect itself from being overwhelmed by information on the channel. There is no inherent limitation on the number or type of filters that may be activated in a channel but implementations will probably constrain this variability.

The filter can be thought of as a gate through which information is required to flow. Information that fails to satisfy the gate's criteria closes the gate. The location at which this filtering occurs is determined by a parameter provided when the filter is activated. In general, filtering is performed most effectively at the source of the information but may be activated at the destination if multiple subscribers prohibits filtering at the source. This decision is made at test planning and is known when the request to activate the filter is issued.

4.5.2.2.4.14 Modify Filter

This service is used to change the value of one or more parameters to a filter that is activated in an information channel.

4.5.2.2.4.15 Deactivate Filter

This service is used by a component that has established an information filter in one of its information channels to deactivate that filter. If there are multiple filters activated in a channel they must be deactivated individually.

4.5.2.2.4.16 Request Update

This service can be used by a system component that has subscribed to some information that is being published by some other system component to query the publisher for the current value of the information described by the parameters. This applies to static classes and not events. This request has optional parameters that can be used to direct the request to a particular asset or apply to a specific class instance. If no optional parameters are used the request is directed to all assets that are publishing the class. For all instances the response to this query is satisfied by publishers issuing updates for the indicated information. These updates are directed to the specific requesters.

4.5.2.2.4.17 Register Channel

This service is used to identify the protocol and circuit that has been allocated as a dedicated public circuit. This service is used to identify the actual circuit descriptor that is to be used in propagating the updates for the published class.

4.5.2.2.4.18 Unregister Channel

This service is used to indicate that the protocol and circuit that has previously been allocated as a dedicated public circuit is no longer required. This service is used to identify the actual circuit descriptor that is to be removed from the internal tables of Distribution Services.

4.5.2.2.4.19 Terminate Asset

This service is used to clean up the tables maintained by Distribution service when an asset has been placed out of service by a user command to Execution service. This is similar to the clean up that is normally performed at the end of an exercise or when an asset removes itself from the exercise. In this case the asset has failed and is not able to participate in the normal termination sequence.

4.5.2.2.5 Interfaces:

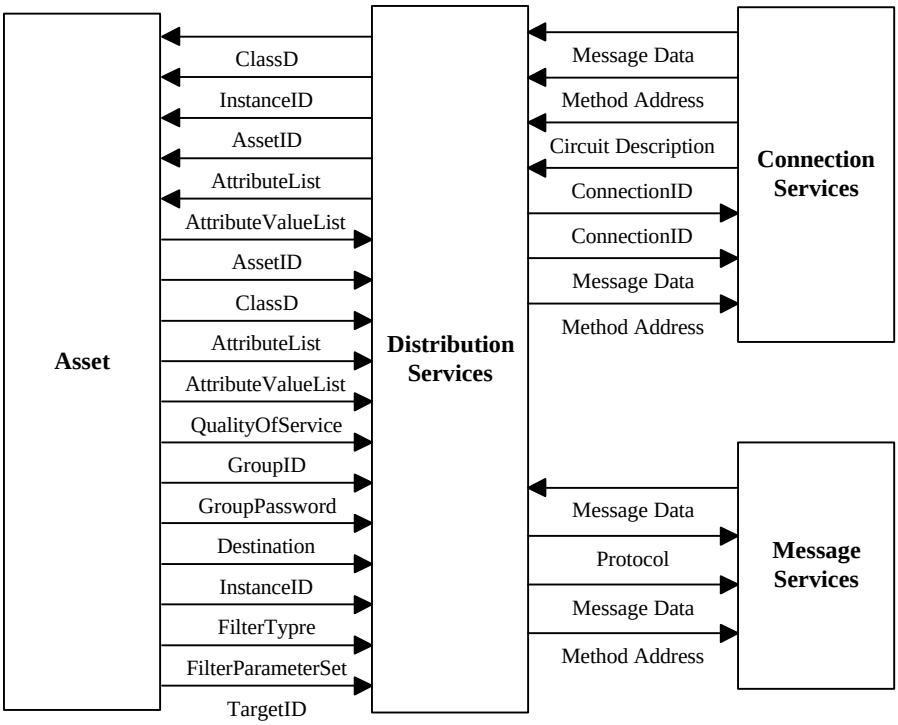

Figure 38. Distribution Service Interfaces

4.5.2.3 Message Services

4.5.2.3.1 Description:

Message services provides a form of communications based on a discrete, packetized data oriented service for any purposes applicable within facility operations. It is expected that all communications for

discrete data streams which do not have exceptional latency or capacity requirements or which are not restricted to specific dedicated channels for other purposes will be managed through message services. It provides connections to one or more networks to support these services by making use of communications facilities provided by the execution platform. Users of message services can select among different networks and protocol sets to match provided capabilities and characteristics with required quality of service. This service functions primarily to encapsulate the interfaces to the network services present on the platform. It isolates users from the details of how protocols operate, what system services must be used, and format and content of control data.

From the users perspective this service is a datagram service. However, the quality of service negotiated may establish virtual circuits where required to ensure meeting guarantees for delivery, restrictions on ordering, etc.

The basic strategy for sending messages is for the service requester to buffer the outgoing message and provide parameter values to determine destination and distribution characteristics. The sender then calls the send service. For incoming messages the service requester can poll for messages by using the receive service, with appropriate parameter values. Implementations may also allow an interrupt driven option that allows message services to call an entry provided by the service requester to deliver the message. With an interrupt driven approach message services will not buffer messages for users. Service requesters can switch between polled and interrupt driven service as required. However, when switching from polled to interrupt driven the service requester is responsible for retrieving any messages remaining on the incoming message queue when the switch is enacted.

4.5.2.3.2 Assumptions

Users of Message Services understand the quality of service which can be guaranteed for each protocol which Message Services provides access to. Users provide any mapping between requested quality of service parameter values and protocols supported here.

4.5.2.3.3 Rationale

The primary drivers for identifying the Message Services partition are separation of concern and information hiding. Both of these are consequences of a need to limit the complexity of Distribution Services by removing the need to deal with the details of how datagrams are sent and which platform facilities to make use of in dispatching packetized data from Distribution Services. The requirement to provide packetized network connectivity is allocated to Message Services to relieve Distribution Services of those details.

In order to provide the capability to select different strategies for managing the servicing of incoming message streams Message Services is provided with the ability to support both synchronous and asynchronous message delivery on a protocol basis. Users (Distribution Services) can alternate between the approaches as required to meet changing needs and better match overhead costs and latency to user requirements.

4.5.2.3.4 Services Provided:

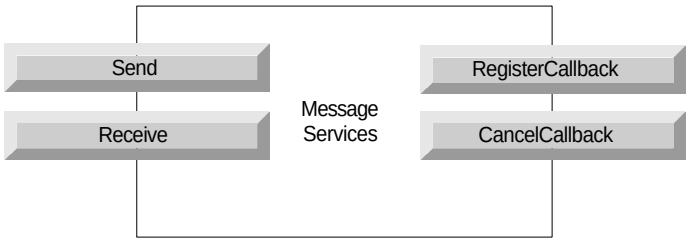

Figure 39. Message Service - Services Provided

4.5.2.3.4.1 Send

A service used to send a message.

4.5.2.3.4.2 Receive

A service used to extract the next message on the incoming message queue for the indicated protocol.

4.5.2.3.4.3 Register Callback

This service is used to register a method for callback service when an incoming message is received. A callback is registered for each protocol set that can be used to transfer messages. Invocation of this service sets the retrieval mode for the protocol to interrupt driven. The mode defaults to polled when the infrastructure is initialized and remains so until this service is invoked.

4.5.2.3.4.4 Cancel Callback

This service is used to cancel a previously registered callback. It removes the callback method address for the protocol indicated and marks the protocol as polled.

4.5.2.3.5 Interfaces:

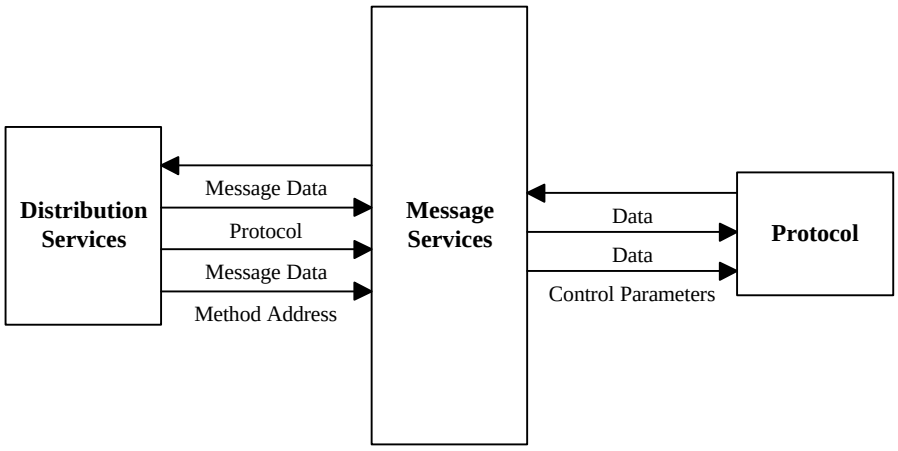

Figure 40. Message Service Interfaces

4.5.2.4 Connection Services

4.5.2.4.1 Description:

Connection Services provides the system mechanisms for dynamically adding and removing communications networks and protocols. The service supported here is similar to that provided by Message Services for well-known platform supported communications networks but allows those communications facilities to be added and removed from the system in response to the needs of specific logical ranges and, further, allows the use of special purpose protocols which support peculiar communications needs or afford improved quality of service tailored to test requirements.

Connection Services accepts requests to install communications connections into the system including arranging requested protocols and connecting them into the data stream. The assembly of protocols and connections is referenced by a communications channel number assigned by Connection Services and the quality of service parameter value set that the connection supports. It advises Distribution Services of the channel it supports along with the quality of service supported by the channel and a method handle used for sending information so that Distribution Services can provide support to publishers and subscribers.

4.5.2.4.2 Assumptions:

Asset Manager has ascertained the accuracy and completeness of information about connections attached to the operating platform.

4.5.2.4.3 Rationale:

Connection Services was created in response to a need to simplify Distribution Services by removing from it the need to deal with the specifics of how to connect up a communications channel. It was also separated from Message Services to allow each service group to optimize their implementation for either statically configured connections and protocols or dynamically configured connections and protocols.

4.5.2.4.4 Services Provided:

The connection services provide two distinct groups of services, network protocol messaging services as are provided using message services and network connection management. The TENA relies on network manager to provide facility specific network management.

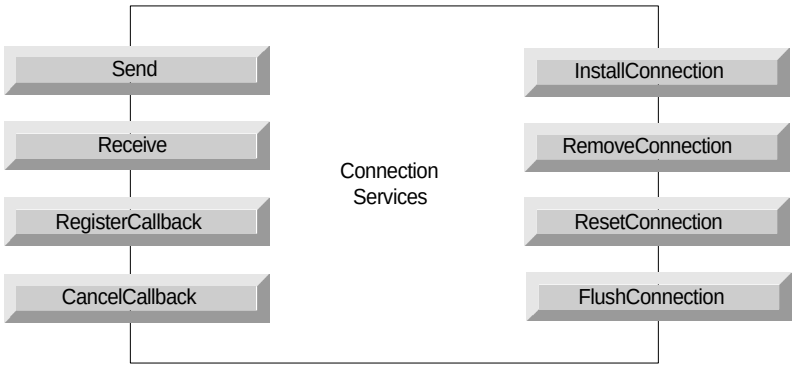

Figure 41. Connection Service - Services Provided

4.5.2.4.4.1 Send

A service used to send a message.

4.5.2.4.4.2 Receive

A service used to extract the next message on the incoming message queue for the indicated connection.

4.5.2.4.4.3 Register Callback

This service is used to register a method for callback service when an incoming message is received. A callback is registered for each connection which can be used to transfer messages. Invocation of this service sets the retrieval mode for the connection to interrupt driven. The mode defaults to polled when the infrastructure is initialized and remains so until this service is invoked.

4.5.2.4.4.4 Cancel Callback

This service is used to cancel a previously registered callback. It removes the callback method address for the connection indicated and marks the connection as polled.

4.5.2.4.4.5 Install Connection

This service is used by infrastructure components to install a communications circuit that has been attached to the operating platform. The Network Manager creates, acquires, or allocates a circuit and attaches to an input/output channel on the local platform. Connection information supplied by the Network Manager about this connection is provided to Connection Services to locate and characterize the circuit.

4.5.2.4.4.6 Remove Connection

This service removes a connection from the set of connections that are supported by Connection Services.

4.5.2.4.4.7 Reset Connection

This service is provided to enable the reinitialization of a connection and associated protocols to recover from errors.

4.5.2.4.4.8 Flush Connection

This service forces all data that has been queued within protocols associated with the connection, within communications nodes, and within network interface units to be delivered to the connection end point. Additional information will not be accepted until all connection data has been “flushed” from buffers and stacks.

4.5.2.4.5 Interfaces

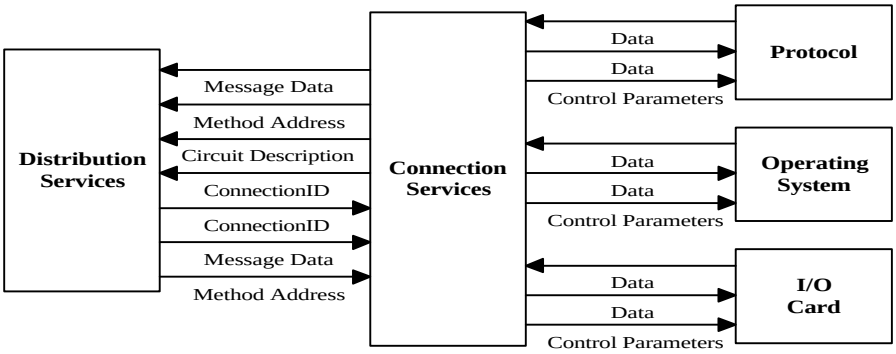

Figure 42. Connection Service Interfaces

4.5.2.5 Clock Services

4.5.2.5.1 Description

The Clock service group manages time issues for the facility. It performs synchronization and time setting services as well as maintaining a global clock for exercises.

4.5.2.5.2 Assumptions

The specification of this services group is based on the following assumptions:

- The infrastructure has a method for reliably maintaining an internal wall clock such as GPS.
- The Clock service group has subscribed and published to all event classes needed by Clock services.
- The variability in network latency times is sufficiently small that time service algorithms can be calibrated to remove the network effects.

4.5.2.5.3 Rationale

Synchronization of time is an important requirement for many TENA assets. Examples include radars and other devices that time stamp data before it is sent to other assets. Without all assets having synchronized clocks, data collected during a test or exercise can be useless.

4.5.2.5.4 Services Provided:

Figure 43. Clock Service - Services Provided

4.5.2.5.4.1 Get Current Time

This service is utilized to get the current value of date and time for the facility system global clock. This is a single synchronized date and time that is shared by all facility instances. It is Zulu time.

4.5.2.5.4.2 Set Current Time

This service is used by a facility management asset with sufficient authority to set the current value of the date and time maintained for the facility system clock. For facilities with assets that use GPS to maintain their clock, the Set Current Time will have no effect. Set Current time is used to set the time for those assets without a GPS capability.

4.5.2.5.4.3 Sync

This service is used to cause the infrastructure to propagate a date and time update message to other infrastructure instances.

4.5.2.5.5 Interfaces:

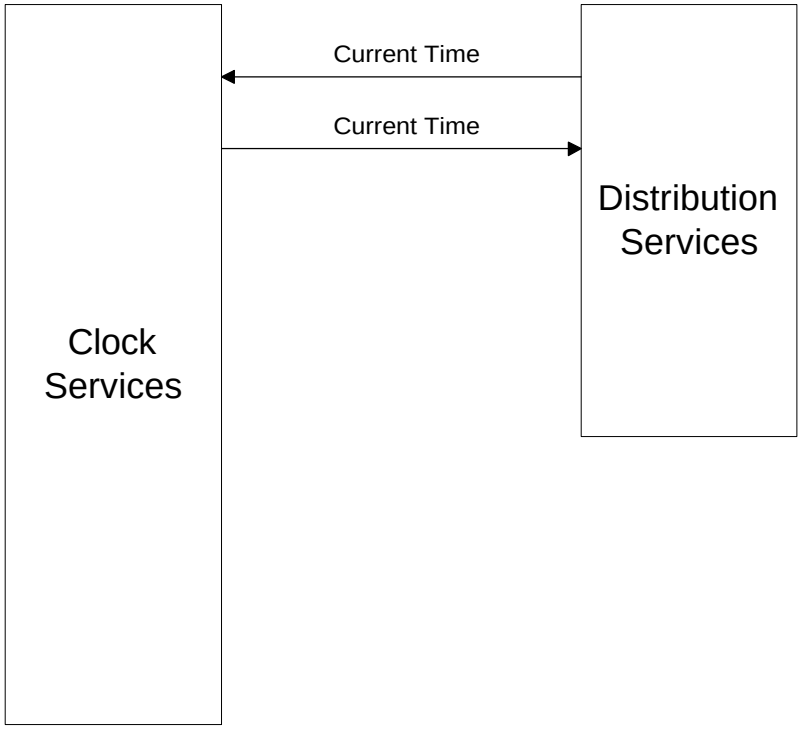

Figure 44. Clock Service Interfaces

4.5.2.6 Infrastructure Support Objects

The infrastructure component labeled Infrastructure Support Objects is intended to contain the definition for and instances of classes of basic objects required for system functioning rather than exercise execution. Many of the classes or objects which appear in the object model will appear as implementations within this group. The set of basic type definitions and associated methods which are required across the system are established here as well as definitions provided for more complex classes which are needed widely or are fundamental to capabilities supported directly by the TENA core. It is also likely that a number of basic utilities will be provided as part of this component for use across the system.

4.5.3 Mandatory Application Programs

A concern when designating a capability as an application is that it implies a certain freedom in the way it will be implemented. This is not the case with the TENA required or mandatory applications. Standardization of implementation is critical for Asset Manager, Execution Manager and Initialization Managers. Network Manager may be implemented to conform to local site configurations and may therefore vary in its implementation. In order to enforce this standardization of mandatory

applications, these applications have been included as part of the TENA Core and will be standard across all instances of the Logical Range.

4.5.3.1 Network Manager

4.5.3.1.1 Description:

The Network Manager (NM) is a required application asset which is responsible for managing use of networks and other communications facilities required for facility operations. It is charged with managing the connection of network elements to form connections, it monitors network and connection performance, and it provides network and connection status to the infrastructure and TENA applications. The NM may be a single stand-alone asset or may be integrated into a larger facility management application depending on requirements at each facility and implementers' preference.

The NM is intended primarily to deal with the dynamic acquisition and return of communications resources to satisfy individual exercise requirements. It manages network resources that are permanently connected to facility assets and supported by the operating platforms they are attached to. However, these permanent networks, which support information transfers managed by Message Services, usually require little management. The bulk of activity is in dealing with circuits and networks intended for dedicated public use managed by Connection Services and for private connections directly connecting two specific assets.

The NM works closely with Asset Manager (AM) to create, acquire, or configure communications resources in response to the definition of logical ranges or in response to unplanned user requirements. The NM has information about the location of and characteristics of communications resources available to a facility. In response to requests from AM to acquire a communications resource with a specified quality of service (QOS) the NM will determine which resource known to it and available most closely matches the QOS requirements. It then performs the necessary actions to acquire control of the resource. This may involve requesting personnel to connect hardware, configure networks through autonomous network control terminals, requesting and negotiating for common carrier resources, etc. When the NM has determined the resource is available for use by the facility it reports the status and actual QOS characteristics, along with the input/output channel connection point and machine, to AM.

During operation the NM monitors all communications resources which it has available to it or knows about and reports this status for use by other facility assets. It may also be augmented with the capability to act to relieve congestion in circuits. AM may also request that a resource be released when it is no longer required. The NM provides capabilities to accomplish this in a timely manner.

The resources managed by the NM are intended to satisfy all facility requirements and not just computer data transfer. The NM may be required to provide and manage resources for the transport of integrated voice, video, and data communications. It may further be required to provide for the transport of raw, compressed and/or secure voice and video communications. The transport must enable real-time reconstruction of the data by the receiving asset and meet network command/control requirements.

The NM must be extensible and flexible so that it can quickly adapt to new communications technologies as they become available and additional or changes requirements for supporting tests and facility business operations.

4.5.3.1.2 Rationale:

Processing power is doubling about every two years. LAN capacity development has not kept pace with this increase in processing power resulting in fewer computers being attached to a LAN segment. To meet the increased networking demands new families of LANs / WANs are evolving.

Networking protocols developed 10 or more years ago placed greater functionality within network components to compensate for reliability and limitations in data links and to off-load end-point systems. These new emerging transport technologies coupled with more reliable transport media have greatly reduced the need for these embedded network services.

Facility communication requirements are viewed as critical design criteria. One of the design goals of the TENA system architecture is to provide the data transport flexibility needed to maximize communication performance and efficiency.

Modern facility systems require the integration of voice, video, and data. To meet these demands and the continuing increase in network bandwidth requirements, emerging transport technologies are greatly reducing the services provided by the network elements and moving them to the customer premises equipment. The TENA infrastructure must support both the current generation of transport services and the emerging networking systems support services. To replace this reduction of operations, administration, and maintenance capabilities, the infrastructure and logical range support applications provide greater support for network management services.

DoD T&E/Training facilities can no longer afford the many leased lines to interconnect facility systems. Many point-to-point leased lines have no backup resulting in reliability problems. Additionally, use of leased lines may be low as lines may be sized according to peak capacity requirements.

A better networking solution to meeting facility communications needs is to reengineer these existing LAN/WAN networks into a new network which provides efficient circuit switching for both backup capabilities and the sharing of expensive lines. This new network architecture allows dynamic capacity allocation to maximize available data bandwidth utilization.

These reengineered telecommunication systems are based on the idea of relaying traffic as quickly as possible using fast packet relay or fast packet switching. The networking technology used to transport data is transparent to the TENA infrastructure. Transparency allows tailoring of networking infrastructure, permitting facilities to use the technologies that best meet local requirements.

4.5.3.1.3 Capabilities

Network Manager services can be grouped into three general categories: circuit routing, circuit service, and circuit status. Circuit routing includes those services needed to determine, create, and configure a connection. Circuit service services include diagnostic and testing functions, traffic and congestion

management, switch over operations, and timing control. Circuit status information pertains to the status of a connection/network and status of network elements (assets).

The network associated with a physical facility may be partitioned into disjoint subsets (which may not necessarily coincide with facility segments). Each network subset has an associated network manager. Each network manager is responsible for the operation, administration, maintenance and provisioning (OAM&P) of its associated network elements.

The OAM&P services provided by TENA network elements allow for traffic control and congestion control. Congestion is defined as a condition that exists at the transport layer in the network elements where a circuit is not able to meet a stated or negotiated level of performance. Traffic control defines a set of actions taken by the network manager to avoid congestion. Traffic control takes measures to adapt to unpredictable fluctuations in traffic flows (information bandwidth) and other problems within the network.

Associated with a connection are certain quality of service attributes. These attributes specify the network parameters of the technologies used to support the connection. The infrastructure and network manager allow for scheduling of network assets with specified QOS parameters and for dynamic redefinition of QOS during a test. These QOS attributes determine what connection admission control actions and circuit performance parameters will be used in network manager.

It is the policy of the TENA not to manage network QOS but to provide the services that allow user assets the ability to manage. QOS management is shared among the user, the network manager, and the network. The mechanism by which this management is carried out is the service contract. The network user is responsible for agreeing to a service contract with the TENA that stipulates the rules on the use of a network (such as bandwidth); circuit performance requirements (such as circuit reliability); and acceptable alternative actions if degraded conditions exist (such as increased error rates resulting in reduced effective data bandwidth). The network assumes the responsibility of supporting the other QOS requirements. The TENA may support contract re-negotiation to enable dynamic network management.

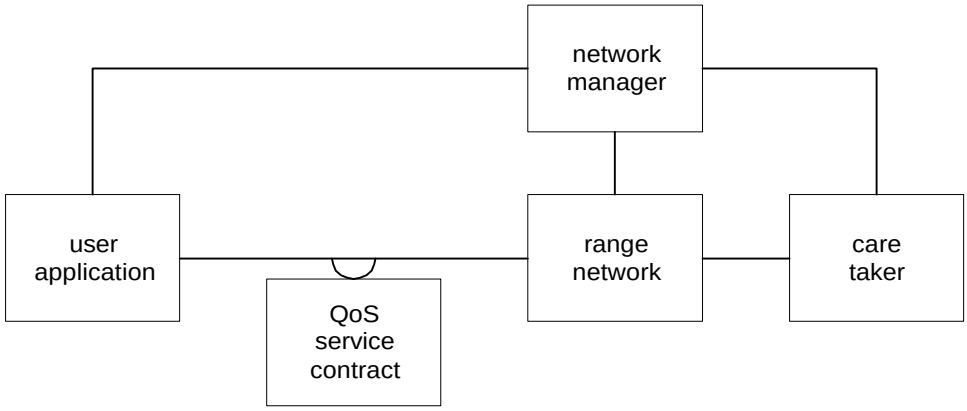

Figure 45. Network Manager Interfaces

QOS parameters are defined in the data dictionary. Any particular facility may support all QOS parameters or only a subset of them. They may include the following:

- Protocol selection - protocol selection applies to message services as well as to connection services.
- Communications latency on a connection.
- Required level of reliability of data delivery. The implementation of delivery reliability may be through the protocol selected (TCP/IP or UDP/IP, for example) and/or through the transport technology used.
- Number of errors tolerable or error rate limit desired.

Additional QOS parameters may be defined by facility system implementers to satisfy local requirements. Each facility will establish a default set of qualities of service which satisfy basic requirements and map to well known, platform supported resources.

The status of network subsets becomes the shared responsibility between caretaker(s) and network manager. Network status is composed of, schedule status and current operating state.

The responsibility for operating, administering, maintaining, and provisioning of facility connections lies with the network manager. The network manager resides outside the infrastructure to allow for tailoring of its services to specific network requirements and special facility operations needs. The network manager, in collaboration with network asset caretakers, is required to support infrastructure scheduling services as well as general network operations support.

Fundamental to the TENA technical reference architecture is the design decision that the route that data takes through a network isn't important as long as the connection's QOS is satisfied. The QOS of a connection makes up one side of the service contract.

The details of the service contract will be determined by the facility development group. However, the procedure used to establish this contract can be described. In general: An analyst or support tool determines the aggregate QOS requirements for all connections needed to support a test. These requirements are translated to numbers of connections with associated quality of service which drive the NM to create connections as commanded by the Asset Manager. The specifications are integrated within the test plan and are, in essence, the service contract specification. During test conduct, these connection specifications are extracted from the test plan by Execution Manager and subsequently passed to AM. AM processes these connection requests and passes the connection information along with QOS specifications to the NM.

4.5.3.2 Asset Manager

4.5.3.2.1 Introduction

Business processes supported by TENA are related to the management of assets that facilities own and make available for general use by members of the enterprise. This is primarily related to the

identification and scheduling of assets for use in supporting tests or exercises and the operation of the logical range instantiated for test or exercise conduct. The test plan is a description of all of the assets required for a test or exercise, the datasets which are required to initialize assets or which are required by assets during execution of the operation, the communication resources to be employed for the test or exercise (including how assets are *wired up*), as well as information concerning how the operation is to be controlled. Test plan information, along with information logged during conduct of the exercise is also available for use in determining costs and generating billing information.

In the discussion below the term user refers to an application that the facility personnel interact with via some appropriate human-computer interface. In some cases this application is a TENA Core application, a mandatory application present in each instance of the TENA Core, and in other cases the application is a facility specific application. The operation of these applications is essentially identical in terms of how they interact with the enterprise and TENA Core although they may be quite different with regard to other aspects of their operation.

4.5.3.2.2 Asset Operating Mode

Every asset defined in the master asset catalog has a mode of operation that effects scheduling of that asset. An asset can be in one of the following modes: Independent, Shared, and Exclusive.

Assets that are available for general use and have no restrictions on the number of simultaneous users or are used on a first come first served basis operate in the Independent mode. When these assets are operational they are available for use by any logical range without restriction and do not require that reservation tokens or access tokens be obtained before they can be used. Users must be cognizant of the fact that these assets may be overloaded when operating in this mode. In general, facility designers will provide adequate resources and sufficient copies of Independent mode assets so that overloading is not a problem.

Assets which operate in the Shared or Exclusive mode are schedulable assets and require that a logical range obtain a reservation token and access token before they can be used. A reservation token is a token provided by the asset's caretaker which affords the requester access to the asset during the reserved period. This allows for planning use of facility assets. It does not, however, guarantee that the reservation will be honored. Any reservation may be overridden by a higher priority user and conditions of exercise execution may cause a test or exercise to run over its reserved time slot for an asset. Assets may also become unavailable because of failures or required maintenance. It is the responsibility of a facility owner to manage the schedule for his facility assets. Facility owners accomplish this through the asset caretakers and management applications programs which monitor and control use of assets and which are provided with sufficient privileges to access status information and issue commands to control asset allocation.

Shared assets allow for simultaneous use of the asset by more than one user. The number allowed is determined by the asset owner who sets the number of reservation tokens and access tokens which are created for the asset. The owner of the asset must insure that when the allotted number of users are making use of the asset that the asset can support the entire user set with acceptable performance. In cases where there is only a single token, hence one user at a time is allowed, the operating mode is Exclusive and the asset is then completely dedicated to the user. For example, instantiable assets which

are single threaded or assets with capacity restrictions which prohibit more than a single user at a time would operate in the Exclusive mode.

4.5.3.2.3 Reserving an Asset

The Protocol for reserving an asset is as follows: A user executes the Reserve Asset service that transmits the request to the caretaker of the asset. The caretaker checks the current asset schedule to determine if the reservation can be honored. If the slot is available then the request is pushed unto the request queue. User applications review this queue at some interval using the Get Reservation Queue service. The facility owner (directly or through an application program on the caretaker) decides whether to grant or deny the request and issues a Process Reservation Request service to Asset Management Services (a TENA Core mandatory application) indicating whether the request is to be satisfied or denied. The caretaker then responds to the requester with the status of the reservation and, if the reservation was granted, a reservation token.

If the requested schedule slot is not available the caretaker responds to the requester with a message denying the reservation. If the requester has set the persistence flag to indicate that he wishes to persist in obtaining the reservation the request is pushed unto the request queue. The asset owner can then decide, when the request queue is reviewed, to either deny the request or grant it by overriding the current reservation holder. Overriding an existing reservation is accomplished by invoking the Relinquish Token service for the current reservation token holder then issuing a Process Reservation Request service call for the new requester. These actions are reflected to the effected users though the local Asset Management Application component of the TENA Core. The basic strategy for a user to obtain an asset reservation would be to initially request the reservation with persistence set to not persistent then if notified that the reservation attempt failed determine if it is necessary to attempt to get the existing reservation overridden, to select a different time slot for the asset, or to schedule a different asset. If the choice is to attempt to override the existing reservation the user would again invoke the Reserve Asset service with the same asset identity and time slot but with persistence set to persistent.

The owner of an asset may use any appropriate criteria or process to adjudicate reservation requests. For example, an owner may want groups of assets to be used collectively and, therefore, scheduled as a group, or owners may require a review of asset requests by a scheduling committee before deciding which requests to grant. The policies may be implemented in a manual fashion or by an application program operating on behalf of the owner. Since the owner determines what collection of equipment, space, personnel, materiel, etc. is an asset for scheduling purposes the owner controls the granularity for scheduling purposes. All of the assets are described in the Master Asset Catalog along with information on their composition and scheduling restrictions.

Some asset reservations may be allocated on the basis of a capacity, for example, bandwidth of a communications channel. Requests for these assets are accompanied with a description of the requested capacity allotment. This allotment request is used to determine how many simultaneous users can be supported and can be a decision criteria for granting the reservation.

4.5.3.2.4 Asset Control.

As stated earlier, the holder of a reservation token is not necessarily guaranteed access to the asset. If an exercise runs over their reservation time they cannot, in general, be halted indiscriminately. They must either be allowed to complete or facility personnel controlling the exercise must decide whether the exercise can be terminated early. These decisions are beyond the concern of the architecture. Assets may also become unavailable because of faults or required calibration, etc.

This set of conditions is managed by requiring that a logical range obtain an access token before the asset can be used (if the asset is not operating in the Independent mode). This token entitles the holder to make actual use of and control the asset. The access token is obtained by a logical range through an invocation of the Acquire Asset service. The caretaker responds by examining the asset's status (equivalently, searching for an access token). If a valid reservation token was provided by the requester and an access token is available, the access is granted and the access token is dispatched to the requester. If access cannot be granted a denial response is sent to the requester. This means that unscheduled use of assets is supported. If conflicts arise for unscheduled use the facility operators must adjudicate them. Users should note that unscheduled access to an asset requires that they first obtain a reservation token, typically set with the time slot to run from the time of request.

The asset owner, through an application program, may override an access request or cancel an access request which has already been granted, even during the conduct of an exercise, by invoking the Relinquish Token service for the appropriate token holder. User programs must honor this command.

Information on the holder of tokens and any asset status can be obtained by using the Query Asset Status service.

4.5.3.2.5 Reservation Cancellation

As described above, a reservation can be canceled at any time by the owner of the asset. This is accomplished though the asset's caretaker and a privileged applications program. The Process Reservation Request service, and Relinquish Token service should be limited to use by the privileged application which acts on behalf of the owner.

No reservations are granted directly by Asset Management Services. All requests are queued and wait for owner action via a Process Reservation Request service. Asset Management Services will remove any request that is not acted upon before the end of its requested schedule time.

At any time the owner may cancel a reservation by issuing a Relinquish Token service to the reservation holder. The reservation holder must surrender the token. If the reservation holder does not or cannot respond Asset Management Services will note the fact and when a request for the access token comes from the canceled holder it will be denied. Asset Management Services will vacate the reservation slot of the asset's schedule for the canceled reservation.

If a cancellation occurs after the reservation holder has obtained an access token the owner must issue a Relinquish Token service request. The holder of the access token must cease use of the asset and surrender the token in the most expeditious manner. However, under certain conditions neither the infrastructure nor any applications assets may be able to force a user to give up use of an asset when canceled. Facility operators may have to intercede to recover the asset.

A reservation request may itself be canceled by the requester by using the Rescind Reservation Request service. This service is only effective for reservation requests issued by the asset attempting to withdraw the request.

4.5.3.2.6 Description

Manages requests for use of assets in the master asset catalog. An exercise is conducted on a logical range. The status of and schedule for assets in the master asset catalog are managed here. Facility caretakers at each facility are queried by Asset Manager for the assets that are physically located at that facility. Assets are reported available or not available when requested by facility management applications.

Scheduling certain assets may involve human interaction to schedule. This needs to be recognized and mechanisms provided to stimulate the agency that effects the scheduling and record pertinent information for subsequent use during exercise execution. Asset Manager works in concert with Execution Manager to resolve these sorts of manual operations or manually assisted operations happen.

Asset Manager is also responsible for managing access to assets during the execution of a test. Scheduling an asset for use during a test does not guarantee that the asset is available at the scheduled time. The asset may be down for maintenance, another user may still be using it, or any of a number of causes could lead to the unavailability of an asset at the previously scheduled time. During the execution of a test, Execution Manager will query Asset Manager when access to an asset is required. Asset Manager will grant access to the asset if the requesting test holds a valid reservation and the asset is available. If access is not granted Execution Manager can query an appropriate facility management application for resolution of the conflict.

Additionally Asset Manager is responsible for the instantiation of instantiable assets that are not currently executing and coordinates with the Network Manager to have dedicated circuits established and allocated.

4.5.3.2.7 Services Provided

Figure 46 . Asset Manager - Services Provided

4.5.3.2.7.1 Reserve Asset

This service is used to request dedicated service over some period of time from a system asset. Asset Manager will assign a unique request ID to the request and place the request on the asset's request queue.

4.5.3.2.7.2 Release Asset

Used by a system component to release an asset that it has previously reserved for use. Returns the reservation token.

4.5.3.2.7.3 Get Schedule

This service is used to obtain the current schedule for an asset or set of assets managed by a caretaker.

4.5.3.2.7.4 Clear Schedule

A service called to mark an asset's schedule as completely open. Only the asset's caretaker can clear its schedule.

4.5.3.2.7.5 Add Asset

This service is used to add a new asset to the system. This includes adding the asset to a caretaker's set of managed assets, modifying the local asset catalog and schedule to reflect the new asset, and adding the new asset to the master asset catalog.

4.5.3.2.7.6 Remove Asset

This service is used to remove an asset from the system. This includes removing the asset from its caretaker's set of managed assets, modifying the local asset catalog and schedule to reflect the asset deletion, and deleting the asset from the master asset catalog. Only an asset's caretaker can remove it.

4.5.3.2.7.7 Modify Asset Property

This service is used to modify the value of an asset's property. These properties are listed in the local and master asset catalogs and used to control use of the asset. Only an assets caretaker can modify its properties.

4.5.3.2.7.8 Query Asset Status

This service provides a means for users to obtain information on the value of an asset's properties.

4.5.3.2.7.9 Acquire Asset

Users invoke this service to obtain use of an asset. That use may be exclusive or shared depending on the asset. Once acquired an asset is available for use by the requester until the asset is relinquished. If the asset is an instantiable asset, Asset Manager is responsible for instantiating it and generating an instance ID. If the asset is a dedicated communications channel Asset Manager is responsible for coordinating with the Network Manager to have the dedicated circuit established and allocated.

4.5.3.2.7.10 Relinquish Asset

This service is used to relinquish an asset and make it available for use by other applications. If the asset is an instantiable asset, then Asset Manager will terminate the instance of the asset. If the asset is a dedicated communications channel Asset Manager will coordinate with the Network Manager to disconnect the dedicated channel.

4.5.3.2.7.11 Relinquish Token

This service is a callback service invoked by Asset Manager on a holder of a reservation token. It is exercised when a scheduling conflict arises from a high priority user requesting to override an existing reservation or allocation for an asset.

4.5.3.2.7.12 Get Reservation Queue

This service is used to extract the current set of pending reservation requests for a particular asset or assets.

4.5.3.2.7.13 Process Reservation

This service is used by a caretaker to command that a pending asset reservation be processed. Processing a request implies that it is either being satisfied or denied. Satisfying a request involves the recording of the requesters reservation and dispatch of the reservation token to the requester. Denying a request implies the dispatch of a null reservation token. In both cases the request is deleted from the request queue and the Reservation Token is delivered to the reservation requester.

4.5.3.2.7.14 Rescind Reservation

This service is used to rescind a pending reservation request. It must be called by the application that originally issued the request. In response to this service request Asset Manager deletes the reservation request from the appropriate reservation request queue.

4.5.3.3 Execution Manager

4.5.3.3.1 Description

Execution Manager (EM) manages the execution of a test plan on a logical range. Based on the set of assets and events defined in the test plan, EM:

- Acquires the assets
- Connects them as planned
- Brings them to an initialized state
- Ensures they are ready to begin execution of a plan
- Shuts them down in an orderly fashion upon completion or termination of the plan.

In the event that the plan must be interrupted, this service manages the pause and resume of a plan. The Execution Manager supports the dynamic replacement of assets in an executing plan. A plan can be validated prior to its actual execution by running the plan in validate mode.

4.5.3.3.2 Assumptions

A (PR) exists and is accessible by Execution Manager. The PR will contain all plans currently known to the infrastructure.

The Execution Manager can execute multiple plans simultaneously as long as the required asset scheduling is achievable.

Acquiring certain assets may involve human interaction. This requires that Execution Manager have mechanisms that identify the protocols for this sort of interaction or human assistance and provides mechanisms for the assets to be secured for use. The status of these assets must be reflected in the

logical range status and means provided for Execution Manager to determine that it has, in fact, all of the resources dedicated to it that it needs.

4.5.3.3.3 Services Provided:

Figure 47. Execution Manager - Services Provided

4.5.3.3.3.1 Run Plan

Facility management applications make use of this service to request that a particular plan be executed. Invoking this service causes the logical range which is defined in the test or exercise plan to be instantiated. This act involves the acquiring of all the assets which make up the logical range, the instantiation of assets where required, and the initialization of the assets. This service is not directly involved in the actual conduct of a test or exercise.

4.5.3.3.3.2 Pause Plan

This service is used to cause a pause to be sent to the facility assets that are constituted for a plan. The semantics of a pause to assets are that the asset is to cease producing changes to its internal state when those changes would result from plan play. Sensors would continue to operate and data display assets would continue to display the information they currently possess.

4.5.3.3.3.3 Resume Plan

This service is used to cause a resume to be sent to all facility assets which have been previously paused for a plan. The assets once again engage in free play and respond to events and data as they are defined to.

4.5.3.3.3.4 Terminate Plan

This service is used to cause the orderly termination of a plan. Assets which must engage in special processing to effect an orderly and error free shutdown are directed to do so. Input/output devices and channels are flushed and shut down, files are closed, and access tokens for assets are returned to their caretakers. Instantiable assets are terminated.

4.5.3.3.3.5 Replace Asset

This service is used to substitute one asset for another in an executing plan.

Run-time modification of a plan is needed to support events such as asset failure. In such an instance, it may be possible to remove the failed asset and replace it with another. This service allows for a temporary substitution of the offending asset for the remainder of the plan's execution or until which time the Test Conductor chooses to remove it from the plan execution.

4.5.3.4 Initialization Manager

4.5.3.4.1 Description:

The Initialization Manager manages assets used to initialize other assets associated with a Test Facility. These assets are typically data sets used by applications to establish an exercise's initial conditions, e.g., terrain databases, environment settings, and start conditions. Initialization assets can also include bootstrap procedures for devices, interactive scripts that require a human-in-the-loop, or any other entity that supports initialization. These assets may or may not be associated with a test plan.

4.5.3.4.2 Rationale

The variety of initialization assets and their distributed location suggests the need for a unified concept and generalized service to access initialization assets. Without this service the applications will be forced to implement access mechanisms for each type. The existence of such a service also maximizes reuse potential since available initialization assets will be made visible to the community at large for use in a particular exercise or as examples for use in a similar context.

4.5.3.4.3 Assumptions

The following assumptions are made regarding Initialization Manager:

Life cycle management (e.g., create, destroy) of initialization assets is outside the scope of the service. This service is focused on managing access to these assets.

Any initialization asset can be included in the master asset catalog. Whether it is actually catalogued is dependent on the creator of the asset. However, any initialization asset referenced in a test plan must be included in the master asset catalog.

Initialization assets have important relationships with other assets that are retained as part of the master asset catalog. For instance, an initialization data set may be created to initialize a certain type of simulation software. It is of no value to another type of asset. These relationships can be defined at the asset level (e.g., test plan T uses initialization data set D) or at the metadata level (e.g., asset type AT uses initialization data set type DT).

The granularity of the initialization asset is equivalent to that of the asset being initialized. For instance, if three assets are required for a test, and each requires initialization data, then the data is partitioned into three separate data sets. This avoids the introduction of composite initialization assets that provide broader initialization than what is required for the asset thereby promoting reuse.

There is a many-to-one relationship between an asset type and associated initialization assets.

A single copy of an initialization asset exists although it may be referenced and used in multiple instances of assets that require that type of initialization.

An initialization asset exists at various states of readiness. These states are termed the asset development cycle. (See Notes section for a detailed explanation).

4.5.3.4.4 Service Context

The collection of entities that Initialization Manager interacts with to perform its processing is:

- Distribution service (DS) - the infrastructure service that manages the distribution of data among facility assets
- Asset Manager (AM)- the required application that manages the scheduling of facility assets.
- Initialization Asset Collection (IAC) - it is assumed that Initialization Manager (IM) will maintain a persistent store of initialization asset information. This persistent collection is given the name of Initialization Asset Collection in this specification.
- Requesting Asset (RA) - any asset (including an infrastructure service) that requests an initialization service.

4.5.3.4.5 Services Provided:

The following services are specified for The Initialization Manager:

- Add Initialization Asset - introduces an initialization asset entry into the IAC
- Remove Initialization Asset - deletes an initialization asset entry from the IAC
- Get Initialization Asset Content - retrieve the content of a specified initialization asset
- Copy Initialization Asset Content - place a copy of the initialization asset contents into another asset specified by the requesting asset
- Modify Initialization Asset Access Profile - change the value of one or more access properties associated with an IA entry in the IAC
- Get Initialization Asset Profile - get a copy of a selected Initialization Asset profile for an Initialization Asset entry
- Release Initialization Asset Content - destroy instances of Initialization Asset content exchange classes

4.5.3.4.5.1 Add Initialization Asset

This service provides for adding an initialization asset entry into the IAC. An initialization asset is added into the IAC after it has been assigned an asset ID and added into the Master Asset Catalog (MAC) through Asset Manager. One of two IA entry types can be added to the IAC. An entry can include both its metadata and content. In this case, a content parameter is supplied that contains a list of ICC static class and their attribute values. An entry can also be a pointer to the actual content. In this case, an access profile is provided.

While Asset Manager provides the capability to add assets to the Master Asset Catalog, there is a need to retain additional information about an initialization asset. For example, the access properties will provide details of where the asset is located and the method of access for subsequent retrieval and copy.

4.5.3.4.5.2 Remove Initialization Asset

Remove an initialization asset entry from the IAC. Note that this service does not delete the initialization asset; it merely makes it unavailable through the infrastructure.

An initialization asset may need to be removed from the IAC for several reasons: the initialization asset has been retired from the facility, or it is no longer a correct data set or procedure.

4.5.3.4.5.3 Get Initialization Asset Profile

Retrieve a copy of a selected profile type from an initialization asset entry

In order to provide tools like catalog browsers, it is necessary to provide a service that can retrieve the profiles contained in an IA entry.

4.5.3.4.5.4 Modify Initialization Asset Access Profile

Assign values to the access properties of an initialization asset

Over the course of an initialization asset's development cycle (see notes section) there is likely to be a need to change or extend values required for access to the asset. For instance, an initialization asset could be moved to a different device or location or its visibility to infrastructure users may be temporarily restricted. Note that only access profile information can be modified with this service; modification of content must be performed outside the infrastructure or through a remove/add combination of services.

4.5.3.4.5.5 Get Initialization Asset Content

Provide the content of an initialization asset to a requester.

This service provides a common mechanism for retrieving initialization data regardless of the location or underlying storage mechanism of the asset. This location can include the IAC.

4.5.3.4.5.6 Copy Initialization Asset Content

Copy an initialization asset to a supplied location.

There are several reasons for applications to acquire a copy of an initialization asset:

- Some initialization data sets may be very large and the test plan may request a preload of the initialization data (e.g., a terrain database)
- Test execution performance may dictate the co-location of an initialization asset with the asset itself
- Anew asset's initialization asset can be derived from an existing initialization asset
- To debug a test plan it may be desirable to load a copy of the initialization data to the debug environment

4.5.3.4.5.7 Release Initialization Asset Content

Destroy all instances of initialization asset content exchange classes for the identified initialization asset

This service provides for the destruction of all content exchange objects instantiated for purposes of retrieving the content of an initialization asset. Typically used by AM when Execution Manager relinquishes an initialization asset.

4.5.4 Recommended Application Programs

4.5.4.1 Applications to Support Management of Logical Time

4.5.4.1.1 Background

Software assets that will be introduced to the TENA environment may have internal mechanisms to handle the management of time. Time management within the TENA infrastructure must recognize and work with these inherent characteristics of the assets.

Some application assets will use a concept of logical time. Logical time is an abstract notion of time that drives a model's computations but that does not necessarily match real-time. Logical time simulations can model events in the future, move faster than real-time clocks, and sometimes move forward and backwards in time. Logical time simulations are commonly used for analysis. Simulations of these types may be of use for stimulating a system under test or for post-processing test results.

A collection of two or more applications is said to be coordinated if their operations are constrained to ensure that causal and temporal relationships are properly followed. Sometimes, coordination can be provided implicitly within the applications, once they are synchronized to a common clock. Depending on the design of the application, however, this approach may be insufficient. In cases where synchronization is insufficient to guarantee causality and correct temporal order, the recommended applications [11] discussed in this section may provide the tools to control the sequence of message processing.

[11] Other implementation options for time management also exist, and should be identified and evaluated as part of the TENA transition plan. Because of the possibility of other implementation approaches, we have made these recommended applications rather than mandatory applications.

4.5.4.1.2 Time Management Considerations

Time management requirements in TENA are driven by three considerations. One is the need to enable users to integrate assets that have diverse (internal) approaches to dealing with time. The second is the need to produce correct results in a distributed environment where network latencies can permute the order of messages arriving from different assets. Finally, and most important, is the need to provide test planners with one of the tools they need to achieve repeatability.

4.5.4.1.3 An Implementation Option

At this time, there are no direct services provided as part of the TENA Core for time management. Instead, it is recommended that all time management for TENA be provided indirectly through the HLA RTI. Those services are described in the High Level Architecture Interface Specification, Version 1.2. One possible implementation is to provide access to the RTI will be through a Bridge. An example bridge configuration is shown in Figure 48:

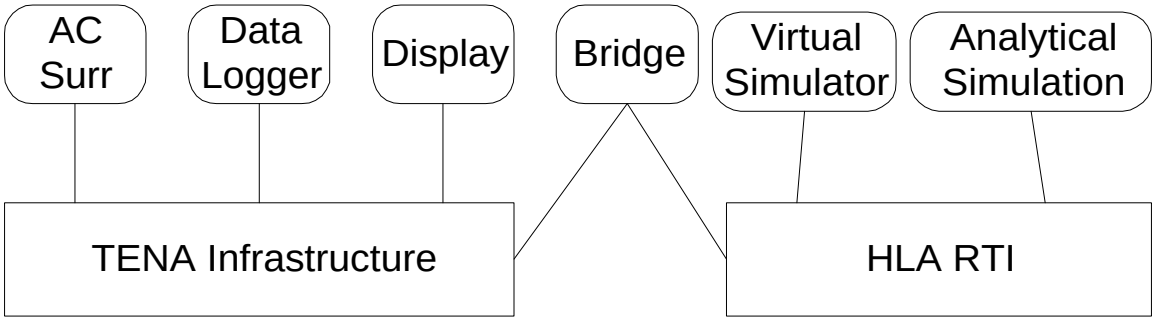

Figure 48. Bridge from TENA to HLA

4.5.4.1.4 Rationale

An analysis of use cases for the TENA domain identified a few assets that require coordinated time management. All of the identified assets are from the modeling and simulation domain. Embedding organic time management services into the TENA would require a substantial effort and may result in performance degradation for all users, even those without a time management requirement. The HLA provides time management services that adequately support the needs of the test and training facilities that use simulations. These HLA services can be effectively exploited through the use of a bridge.

4.6 Relationship of TENA to HLA

4.6.1 Introduction

An important goal of TENA is to reuse the results of other DoD architecture development efforts, when appropriate. The DoD High Level Architecture (HLA) was being specified and developed during the time that the TENA project was initiated. Similarities in the goals of the programs were

obvious. Furthermore, since TENA is required to support training and to incorporate simulations, there was good reason to believe that some of the solutions derived for the simulation community would be appropriate for the test ranges as well. The results of the HLA Engineering Protofederation during fiscal year 1996 confirmed the promise of HLA for test and training range applications, but also highlighted some important HLA challenges that remain ahead, particularly dealing with performance of the Runtime Infrastructure. Furthermore, briefings presented by TENA team during fiscal year 1996 posed some additional concerns regarding the application of HLA to the test and training ranges.

Much has happened since those early days. The HLA has continued to grow and evolve, and a TENA specification has been derived. It's now time to revisit the question of how HLA can be leveraged by the test and training community to evolve to a common architecture.

First, we should note that the TENA architecture was not limited to the views and concepts included in the HLA. Instead, the team based the derivation of the TENA architecture on early results of the requirements analysis and business process modeling tasks. Several members of the TENA IPT team attended technical meetings on the HLA and tracked the evolution of that architecture. It's accurate to say that the HLA work influenced the thinking of the TENA architecture IPT team, but did not constrain its solutions.

The TENA architecture team was charged with developing an architecture to support a business enterprise. The requirements included the need to manage resources and integrate a wide variety of components related to all aspects of operating T&E and training facilities. This was well beyond the scope of allowing components to exchange data. The current concept for the HLA addresses a much more restricted set of issues. It provides a set of basic capabilities which allows components (federates) to be constituted into an execution, supplies some fundamental simulation functionality (save, restore, pause, resume, etc.), and enables the federates to exchange data. Thus, while the HLA may provide essential functionality which TENA can make use of it falls well short of the type and amount of support that TENA needs to provide to the systems it supports. We should point out that this is by no means a shortcoming of the HLA since it has never been the intent of the HLA to support this wide range of system capabilities.

4.6.2 Overall Architecture Similarities

At the top levels, the HLA and TENA share many common features. The functional views partition the system elements into applications and infrastructures. The HLA and TENA infrastructures encapsulate the distributed systems services that provide basic integration and data exchange mechanisms. Applications coordinate their activities with other applications through those infrastructures. These applications are required to interact with their infrastructures using defined interface specifications and Application Program Interfaces (APIs). Otherwise, application designers are given broad flexibility to create applications to suit user need, and the architecture minimizes constraints on the designs. Both the HLA and TENA are intended to support reconfiguration of applications. However, TENA supports a much more robust notion of reconfiguration that provides not only more degrees of freedom to reconfigure but supports reconfiguration over very short periods of time (during exercise execution). This is driven by the transient and rapidly changing nature of T&E/open air training exercises and reliability concerns. Integration can occur before, during or after a test or training exercise.

The TENA and HLA approaches depend on communities of users to achieve data interoperability through agreements on object-based representations of shared domain data. The architectures support two basic kinds of shared information. Static classes have attributes that will typically change status several times over the life of a test or training exercise. These classes have persistence. The architectures also support the sharing of transient data, called interactions in the HLA and events in the TENA. Transient data lasts until delivered to the subscribers of the data.

Differences between the two architectures begin to appear at the infrastructure design levels. In this analysis, we focus on the functions supported by the infrastructures, and the service groups that provide those capabilities.

4.6.3 HLA and TENA Service Groups

The HLA Runtime Infrastructure contains six major service groups. The primary functions of the service groups are listed here:

- Federation Management: provides services for creation, dynamic control, modification, and deletion of a federation execution
- Declaration Management: provides services for publication and subscription of object state information and the interactions generated and received by a federate
- Object Management: provides services for registration, modification, and deletion of objects and for sending and receiving interactions
- Ownership Management: provides services that allow federates to transfer ownership of object attributes
- Time Management: provides mechanisms for controlling the advancement of each federate along the federation time axis
- Data Distribution Management: provides services which facilitate the explicit management of data distribution through a concept known as routing spaces
-
- In TENA the information management infrastructure services are collected into the following groups:
- Distribution Services: Manages the distribution of data between range assets and infrastructure components
- Connection Services: Manages the transfer of information between range assets using information channels (circuits and protocols) that support special quality of service requirements and allow dynamic acquisition and configuration of communications resources
- Message Services: Manages the communications based on a discrete, datagram oriented service based on predefined communications resources supported directly by the underlying platform for any purpose applicable within range operations

- Clock Services: Manages and synchronizes the range system global clock
- Required core applications that support and compliment the information management services are:
- Asset Manager: Manages facility assets including scheduling their use and tracking their status
- Execution Manager: Manages the instantiation of the logical range, setup of assets to support tests and exercises, and execution of an exercise
- Initialization Manager: Manages assets used to initialize other assets associated with a logical range
- Network Manager: is responsible for acquisition of communications resources, configuration of communications channels, monitoring status of communications, and providing for communications reliability

4.6.4 Infrastructure Service Similarities

Some functions are supported by both the HLA and TENA. For example, both architectures use a publication/subscription model for data distribution. This is a major similarity since data distribution is a dominant function of both architectures. The basic capabilities of publication, subscription, object registration, object discovery, and data update are available in both architectures. In the HLA, declaration management provides services for the publication and subscription of data. Object management addresses the management of object instances and updates. In the TENA, distribution services addresses data publication and subscription and the management of object instances and updates.

The architectures use identical approaches for temporal coordination, when needed. In the HLA, time management provides the services that allow federates to coordinate their time and to manage logical time at real-time, slower than real-time, or faster than real-time. Some TENA tests don't need to use this service. However, test and training exercises that include simulations are likely to need this capability. In the TENA, any assets that require coordination of time (other than wall clock synchronization) must use the HLA. One implementation approach is to use an HLA-TENA bridge asset as an interface between the HLA Run Time Infrastructure (RTI) and the TENA infrastructure.

Some functions are supported in both the HLA and TENA infrastructures, but their capabilities are defined differently because of differences in the community requirements.

For example, the HLA and TENA architectures provide support for overall management of tests or training exercises. In the HLA, federation management provides services to create a federation execution and to allow federates to join the execution. Additional services provide for saving the state of the execution, pausing the execution, and resuming the execution. In TENA, management functions such as these are allocated among several required core applications and certain site specific applications. The Execution Manager provides the support for running the test or exercise from either a pre-stored plan or a manually created, unplanned configuration, and for pausing, resuming, resetting, and terminating. Differences between the two approaches are due to the following factors:

- In the TENA domain, test plans are usually, but not always, generated prior to a test. Making use of this plan can make setup more efficient. It also assures us that the plan is followed, or that the TENA infrastructure knows about any deviations from the plan.

- There are a number of safety considerations which must be managed. Many of these require very high reliability support, exceptional system integrity, and direct human control.

- Most of the management for federations is delegated to federates (one or more of which could be exclusively dedicated to this). TENA needs to insure uniform management practices across facilities distributed over wide geographic areas, across all tests and exercises, and for a continuous and extended time period.

- In the HLA, data distribution management (DDM) is used to limit data exchange to defined publication/subscription regions. This enables a federation to reduce the amount of unnecessary data transfers. The RTI does not filter data by region. This is a federate responsibility. The HLA approach to data distribution management was being developed and tested under the STOW program while the TENA architecture was defined. It was too early to assess the effectiveness or efficiency of the HLA approach. An initial assessment of TENA testing applications indicated that typical test range filtering requirements are far more simple than the HLA STOW requirements. Furthermore, the HLA DDM approach assumes that the federates (simulations) contain the logic to perform data manipulation and filtering. This approach would place unrealistic burdens on TENA assets. The TENA approach embeds simple, standard filtering algorithms within distribution services. Efficiency is very important in the real-time testing domain and it is likely that a larger number of filter types is required. TENA allows additional filter types to be added to the system in a relatively straightforward fashion. When more complex filtering approaches (or filters performing transformations on data) are needed by test ranges, users can incorporate these through special filtering assets.

4.6.5 Significant Infrastructure Service Differences

Some functions were introduced into the TENA infrastructure that were not supported by the HLA RTI. One capability that is supported by the HLA RTI was dropped from the TENA infrastructure. This section will describe these significant infrastructure service differences.

In the HLA, the quality of service for the network is an RTI implementation issue. If available RTI implementations do not offer the quality of service needed by users, federation developers are expected to build an RTI that does meet requirements, or to develop an RTI that uses special network channels. In the TENA, the quality of service can be requested at the time of publication. The infrastructure verifies that the network bandwidth is available to support the requested quality of service. This extra flexibility is needed in TENA because of the real-time performance requirements and the need for test repeatability.

In the HLA, the packetizing of data by the RTI is left as an implementation issue for RTI developers. In the TENA, message services packetize data for use in most data transfers that do not require special rates or bandwidths, and connection services ensures that a user-specified quality of service can be provided, even if not anticipated ahead of time. TENA offers more infrastructure support than the HLA for controlling the packetizing of data and providing various qualities of service from communications resources. These services are expected to be important for controlling the performance of high-bandwidth data transfers (e. g. telemetry and video).

TENA provides more flexible, finer grain mechanisms for controlling access to information and insuring secure communications. The T&E and open air range training community may control access to information for more than just restricting access to classified information. They protect information

of a business nature and use the mechanisms to control information in assets which are shared across multiple tests or exercises. The current concepts for information protection in the HLA rely on segmenting executions and separating them with bridge federates which provide guard technology. All federates within a segment operate at the same security level.

In the HLA, infrastructure services are not needed to set a federate's wall clock. The relationship between wall clock time and federate time is managed by the federate. In the TENA infrastructure, clock services are used to synchronize the wall clocks of the various infrastructure instances and their assets. This is required so that exercise results can be interpreted properly and real-time position data can be properly correlated.

In the HLA RTI, no services are defined for scheduling or checking the availability of federates. In the TENA infrastructure, the Asset Manager is used to schedule limited resources. The need for the Asset Manager in TENA is driven by these two considerations:

Some assets cannot be shared during a test; others offer only limited sharing. Careful planning and scheduling is vital to the smooth operation of the logical range. HLA federates, in contrast, are software modules that can generally be copied for other users.

To support the test range business processes (e. g. planning, billing, etc.), information needs to be logged concerning when assets are scheduled and used. The distributed simulation community does not have the same requirements for business process support within the infrastructure.

In the HLA, federate initialization data that is not part of the Federation Object Model (FOM) is not managed by the infrastructure. In TENA, all initialization data for assets are stored and retrieved through the Initialization Manager. The Initialization Manager within the TENA Core encapsulate the access methods and provide opportunities for recording the test conditions. This is useful for interpreting test results and for repeating the test, when necessary. It permits distributed storage of large data sets and transparent access regardless of location of storage or the type of storage technology.

In both the HLA and TENA, the privilege for changing the value of an attribute is uniquely held by a singe application at any point in time during a test or exercise. In the HLA, a federate that has this privilege to update values is said to own the attribute. The RTI provides services to allow federates to exchange ownership of attributes and also to transfer permissions to delete objects. This capability is provided under the ownership management service group. In TENA, the need for capabilities to transfer ownership is envisioned for dealing with reliability and associated issues. We have not determined the form of that support at this time.

4.6.6 Function Mappings to Service Groups

The following table summarizes the functions supported by the TENA and HLA infrastructures and maps those functions onto the service groups or required applications.

Table 1. Summary of TENA and HLA Functions

FUNCTION	COMPARISON	HLA Service Group	TENA Core
Data Distribution	Same functions supported	Declaration Management Object Management	Information Management Services
Temporal Coordination	same functions supported - same implementation recommended	Time Management	Recommended Applications for Logical Time Management
Overall Management	similar -- TENA makes use of test plan and has stop button	Federation Management	Mandatory Core Applications and some site specific applications
Data Distribution Control	similar -- TENA simplifies for asset developers	Data Distribution Management	Distribution Services Filtering Assets
Message Packetizing	similar -- TENA provides greater bandwidth controls	RTI Implementation	Connection Services Message Services
Quality of Service	HLA provides limited implementation dependent choices, TENA allows wide range of dynamic and unplanned choices	RTI implementation	Distribution Services in concert with Connection Services and the Network Manager
Clock Synchronization	TENA only	---	Clock Services
Asset Management	TENA only	---	Asset Manager
Initialization of Assets	TENA only	---	Initialization Manager
Change of instance and attribute ownership	HLA and TENA	Ownership Management	Required but exact approach not determined yet

5.1 INTRODUCTION

5.1.1 Purpose

The section presents the candidate Standards and Protocols for TENA.

The Standards and Protocols provide for agreements on issues that support architectural requirements. These involve data representation, communications protocols, supporting platform capabilities, and processes. TENA standards activity has been focused on identification of available standards. Selection has been deferred to development of system architectures and implementations. Experiments and prototypes will allow us to evaluate candidate selections.

Communications standards are a key area of concern for TENA. TENA has supported a modest test of the applicability of ATM technologies, specifically between Edwards Air Force Base and China Lake. A report of this is provided in Volume X. Additional efforts to categorize representations of open air range data, transfer characteristics, and potential standards is also under way at NAWC, China Lake. Results will be incorporated into a revision of this document.

5.2 IDENTIFIED AND CATALOGUED STANDARDS AND PROTOCOLS

The Standards and Protocols presented in Tables 2-3 were reported as in-use Standards and Protocols by various Range activities. Those presented in Tables 5-16 are Standards and Protocols identified in the Technical Architecture Framework for Information Management (TAFIM) and the Joint Technical Architecture (JTA). Finally, Table 17 lists Range Commanders Council Standards and Protocol Source Documents. This list is provided to support future Standards and Protocol selection and development efforts. The list contains documents that should be good sources for Standards and Protocols or serve as reference material in the development of new Standards and Protocols.

Table 2. Atlantic Fleet Weapons Training Facility (AFWTF) Digital Data Standards

AFTWTF Standards and Protocols		**TENA Categorization**			
Data Standards	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Digital Data Standards	MIL-STD-1397B (SHIPS), Military Standard, Input/Output Interfaces, Standard Digital Data, Navy Systems of 3 March 1980	√			
	Inter-Range Instrumentation Group (IRIG) Time Format B		√		
	IEEE Standard 802.3 (the Ethernet Standard)		√		
	Naval Tactical Data Systems, Model 4, Link-11 Operational Specification, Revision 2 (OS-4-11.2)		√		

Table 3. Miscellaneous Standards and Protocols

Miscellaneous Standards and Protocols		**TENA Categorization**			
Simulation	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Simulation Data Exchange Standards	Distributed Interactive Simulation (DIS) standards version 2.03	√			
	Distributed Interactive Simulation (DIS) standards version 2.04	√			
Networking Protocol	UDP/IP	√			
Communication	DR-19	√			
Advance Range Telemetry (ARTM)					
Telemetry	IRIG-106	√	√		
	IRIG-118	√	√		

Table 4. Communication Services Relationship to TENA

TAFIM Categorization		**TENA Relationship**			
MAJOR SERVICE AREA: COMMUNICATIONS SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Application services					
File transfer	MIL-STD-2045-17504 (FTP)	√			
Remote file access	OSF DCE 1.1: DFS	√			
Message	ANSI/IEEE 1224.1 (X.400 E-mail API)	√			
transfer	ACP 123	√			
(Complementary)	ACP 123 US SUPP-1	√			
Terminal emulation	MIL-STD-2045-17506 (Remote Login Profile)	√			
Remote login	MIL-STD-2045-17506 (Remote Login Profile)	√			
Remote procedure call	OSF DCE 1.1: RPC	√			
Directory services	ITU-T X.500/01/09/11/18/19/20/21/25	√			
(Complementary)	ANSI/IEEE 1224.2 (Directory/Name Space API)	√			
	ISO 8822, 8823, 8326, 8327	√			
	MIL-STD-2045-17505 (DNS) (legacy systems)	√			

Table 4. Communication Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: COMMUNICATIONS SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Addressing (Alternative)	ITU-T X.500:1993 (OSI Directory (ISO 9594))	√			
	ISO 8823, 8327	√			
	IEEE 802.2 (1992)	√			
	MIL-STD-2045-14502-1A/4/5 (Internet Transport Profile)	√			
Protocol for interoperability in heterogeneous transaction processing systems	ISO 10026-1, 2,3:1992 (OSI Distributed Transaction Processing)	√			
Connection establishment/ release (Alternative)	ISO 8823, 8327	√			
	MIL-STD-2045-14502-1A/2/3 (Internet Transport Profile)	√			
	X/Open C303 (XAP)	√			
	IEEE P1003.1g (POSIX protocol - Independent Tranport Service)	√			
	MIL-STD-2045-14503 (RFC 1006)	√			
Connectionless service (Alternative)	ISO 9576/9548 (Connectionless Presentation/Session Protocol)	√			
	MIL-STD-2045-14502-1A/4 (Internet Transport Profile)	√			
	IEEE P1003.1g (POSIX protocol - Independent Tranport Service)	√			
	IEEE 802.2 Type I (1992)	√			
Translation (Alternative)	RFC 1327/1495 (SMTP to X.400 gateway)	√			
	MIL-STD-187-700A	√			
Transport services					
Routing/Relay	MIL-STD-2045-13501		√		
Network gateways	MIL-STD-188-105 (per MIL-STD-187-700A)		√		
Network error recovery	MIL-STD-2045-14502-1A/2/3 (Internet Transport Profile)		√		
Network flow control	MIL-STD-2045-14502-1A/2/3 (Internet Transport Profile)		√		
Network sequencing	MIL-STD-2045-14502-1A/2/3 (Internet Transport Profile)		√		
Priority/ precedence	MIL-STD-2045-14502-1A (Internet Transport Profile)		√		
Distributed timing service	OSF DCE 1.1		√		

Table 4. Communication Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: COMMUNICATIONS SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Multicast (Alternative)	ITU-T X.6 (Multicast)		√		
	MIL-STD-2045-14502-1A (Internet Transport Profile)		√		
Subnetwork technologies					
CSMA/CD (Alternative)	MIL-STD-187-700A	√			
	MIL-STD-2045-14502-4/5 (Internet Transport Profile)	√			
Token bus	MIL-STD-187-700A	√			
Token ring	MIL-STD-187-700A	√			
Distributed queue dual bus (DQDB)	MIL-STD-187-700A	√			
FDDI (Fiber optic)	MIL-STD-187-700A	√			
Integrated services digital networks (ISDN)	MIL-STD-187-700A	√			
LAPB	MIL-STD-2045-14502-2 (Internet Transport Profile)	√			
DDN X.25	MIL-STD-2045-14502-3 (Internet Transport Profile)	√			
Frame relay	MIL-STD-187-700A	√			
Asynchronous transfer mode (ATM)	MIL-STD-187-700A	√			
Combat net radio digital subnetwork (Complementary)	MIL-STD-188-220A (Digital Message Transfer Device (DMTD)	√			
	MIL-STD-2045-14502-6A (Internet Transport Profile)	√			
Secondary imagery transmission	MIL-STD-2045-44500	√			

Table 5 – Distributed Computing Services Relationship to TENA

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: DISTRIBUTED COMPUTING SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Client/server					
Threads (Alternative)	IEEE 1003.1c (Threads Extension to POSIX)	√			
	OSF DCE 1.1: Threads	√			
Remote procedure call	OSF DCE 1.1: RPC	√			
Distributed file service	OSF DCE 1.1: DFS	√			
Naming services	OSF DCE 1.1: Cell Directory Service / Global Directory Service	√			
Distributed timing service	OSF DCE 1.1: DTS	√			
Object services					
Object request broker	OMG CORBA 2.0		√		
Remote access					
File transfer	MIL-STD-2045-17504 (FTP)	√			
Remote login	MIL-STD-2045-17506 (Remote Login Profile)	√			
Remote data access	ISO/IEC 9579-1,2:1993 (RDA)	√			

Table 6. Data Interchange Services Relationship to TENA

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: DATA INTERCHANGE SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Characters and symbols					
Font information exchange	ISO 9541-1,2,3:1991-94 (Font Information Interchange	√			
Hardware applications					
External data representation	ITU-T X.409 (XDR for use with X.400)	√			
Circuit design data exchange	NIST FIPS PUB 172 (VHDL)	√			
Bar coding	MIL-STD-1189B (Standard DoD Bar Code Symbology)	√			
Physical interface (Alternative)	NIST FIPS PUB 22-1 (Synchronous Signalling Rates between Data Terminal and Data Communication Equipment)	√			
	NIST FIPS PUB 100-1 (DTE/DCE Interface)	√			
	NIST FIPS PUB 166 (4800/9600 bps 2-wire duplex modems)	√			
	NIST FIPS PUB 167 (9600 bps four-wire duplex modems)	√			
	NIST FIPS PUB 168 (12000/14400 bps 4-wire duplex modem)	√			
	NIST FIPS PUB 169 (Error correction in modems)	√			
	NIST FIPS PUB 170 (Data compression in V.42 modems)	√			
	PCMCIA PC Card Standard, Release 2.1	√			
Optical digital technologies					
Read-only optical discs	ISO 9660:1988 (Volume/file structure for CD-ROM)	√			
Write-once optical discs (Complementary by size)	ISO/IEC 9171-1:1990 (Unrecorded 130mm WORM)	√			
	ISO/IEC 9171-2:1990 (Recording format for 130mm WORM)	√			
	ANSI X3.191-1991 (130mm WORM)	√			
	ANSI X3.211-1992 (130mm WORM)	√			
	ANSI X3.214-1992 (130mm WORM)	√			
	ISO/IEC 11560:1992 (130mm WORM using Magneto-Optical Effect)	√			
	ANSI X3.220-1992 (130mm WORM using Magneto-Optical Effect)	√			

Table 6. Data Interchange Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: DATA INTERCHANGE SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
	ISO/IEC 10885:1993 (356mm WORM)	√			
	ANSI X3.200-1992 (356mm WORM)	√			
Rewritable optical discs (Complementary by size)	ISO 10900:1992 (90mm Optical Disk, Rewritable and Read Only)	√			
	ISO 10089:1991 (130mm Rewritable Optical Disk)	√			
	ANSI X3.212-1992 (130mm Rewritable Optical Disk Using Magneto-Optical Effect)	√			
Document interchange					
Document exchange (Alternative)	MIL-PRF-28001 (CALS SGML)	√			
	NIST FIPS PUB 152 (SGML)	√			
Custom definition of document types	NIST FIPS PUB 152 (SGML)	√			
Electronic forms interchange	JIEO-E-2300 (Electronic Forms Systems)	√			
Technical data interchange					
Vector graphics data interchange (Alternative)	MIL-PRF-28000 (CALS IGES)	√			
	NIST FIPS PUB 177 (IGES)	√			
	MIL-PRF-28003 (CALS CGM)	√			
	MIL-STD-2301A (NITFS CGM)	√			
	NIST FIPS PUB 128-1 (CGM)	√			
Product data nterchange (Alternative on CALS)	MIL-PRF-28000 (CALS IGES)	√			
	NIST FIPS PUB 177 (IGES)	√			
	ISO/IEC 10303:1994 (STEP)	√			
	MIL-STD-1840B (Automated Interchange of Technical Information (CALS))	√			
Business data interchange	NIST FIPS PUB 161-1 (EDI)	√			
Raster/image data interchange					
Raster data interchange (Alternative)	MIL-PRF-28002 (CALS Raster)	√			
	NIST FIPS PUB 150 (Group 4 Facsimile)	√			
	NIST FIPS PUB 158-1 (X-Windows, for BDF)	√			
Image data Interchange Complementary)	MIL-STD-2500A (NITFS, v. 2.0)	√			
	MIL-HDBK-1300A (NITFS)	√			

Table 6. Data Interchange Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: DATA INTERCHANGE SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
DoD applications					
Military logistics and document Support (Alternative)	MIL-STD-1840B (Automated Interchange of Technical Information (CALS)	√			
	MIL-STD-498 (Software Development and Documentation)	√			
	MIL-STD-1388-2B (LSA Record)	√			
Geospatial data exchange (Alternative)	MIL-STD-2407 (Vector Product Format)	√			
	MIL-STD-2401 (World Geodetic System)	√			
	STANAG 3809 (Digital Terrain Elevation Data)	√			
	STANAG 7074 (Digital Geographic Information Exchange Standard (DIGEST))	√			
	NIST FIPS PUB 173-1 (Spatial Data Transfer Standard)	√			
	MIL-STD-2411 (Raster Product Format)	√			
Symbology graphics	MIL-STD-2525 (Common Warfighting Symbology)	√			
Alternative)	MIL-STD-2402 (Symbology Standard)	√			
	WMO Document #49 (Meteorological Services)	√			
	MIL-STD-1295A (Helicopter Cockpit Display Symbology)	√			
	MIL-STD-1787B (Aircraft Display Symbology)	√			
Exchange of formatted military Messages (Alternative)	Interim MIL-STD-6040 and CJCSM 6120.05 (MTF)		√		
	STANAG 5500 and ADATP 3 (MTF)		√		
	MIL-STD-6011 (TADIL A and B)		√		
	MIL-STD-6004 (TADIL C)		√		
	STANAG 5501 and ADATP 31 (Link 11)		√		
	STANAG 5504 and ADATP 4 (Link 4)		√		
	STANAG 5511 and ADATP 11 (Link 11 and 11B)		√		
	STANAG 5516 and ADATP 16 (Link 16)		√		

Table 6 Data Interchange Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: DATA INTERCHANGE SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
	STANAG 5601 and ADATP 12 (Ship-Shore-Ship Buffer)		√		
	MIL-STD-2500A (NITFS, v. 2.0)		√		
	Joint Pub 3-56.20 through 23 (Multi-TADIL Operating Procedure)		√		
	JIEO Multi-TADIL Data Extraction/Reduction Guide		√		
	JTIDS TIDP-TE (TADIL J)		√		
	Interim JTIDS Message Specification (IJMS) Decision Paper 4 and 5		√		
	IJMS Decision Paper 6 (IJMS SOP)		√		
	MIL-STD-6013 (ATDL-1)		√		
	Variable Message Format (VMF) TIDP-TE		√		
Tactical communications (Alternative)	MIL-STD-2045-44500 (TACO2 for the NITFS)		√		
	MIL-STD-188-203A-1 (TADIL A)		√		
	MIL-STD-188-212 (TADIL B)		√		
	MIL-STD-188-203-3 (TADIL C)		√		
	MIL-STD-188-220 (Digital Message Transfer Device (DMTD)		√		
Continuous Acquisition and Life-Cycle Support (CALS) (Complementary)	MIL-STD-1840B (Automated Interchange of Technical Information (CALS)		√		
	MIL-HDBK-59B (CALS Implementation Guide)	√			
	MIL-M-87268 (IETM General)	√			
	MIL-D-87269 (Database Revisable IETM)	√			
	MIL-Q-87270 (IETM Quality Assurance)	√			
	MIL-STD-974 (Contractor Integrated Technical Information Service - CITIS)	√			
Compression					
Text and data compression	X/Open C436:1994 (Commands and Utilities)	√			
Still image compression (Alternative)	NIST FIPS PUB 147 (Group 3 Compression)	√			
	NIST FIPS PUB 148 (General Facsimile)	√			
	NIST FIPS PUB 150 (Group 4 Facsimile)	√			

Table 6 – Data Interchange Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: DATA INTERCHANGE SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
	ITU-T T.4-1988 (Group 3 Compression)	√			
	ITU-T T.6-1988 (Group 4 Compression)	√			
	ITU-T T.81-1993 (JPEG)	√			
	MIL-STD-188-196 (NITFS Bi-Level)	√			
	MIL-STD-188-197A (NITFS ARIDPCM)	√			
	MIL-STD-188-198A (NITFS JPEG)	√			
	MIL-STD-188-199 (NITFS Vector Quantization)	√			
	ISO/IEC 10918-1 (JPEG)	√			
Motion image compression	ISO 11172-1,2,3:1993 (MPEG)	√			

Table 7– Data Management Services Relationship to TENA

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: DATA MANAGEMENT SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Database management system					
Basic database services (Complementary)	NIST FIPS PUB 127-2 (SQL)	√			
	NIST FIPS PUB 193 (SQL Environments)	√			
ndex sequential access (Complementary)	X/Open D010:1990 (ISAM Developers' Specification)	√			
	X/Open C215:1992 (Data Management, Issue 3: ISAM)	√			
Multidatabase APIs	X/Open P303:1993 (SAG Call Level Interface)	√			
Database administration	DoDD 8320.1 (DoD Data Administration)	√			
Electronic forms	JIEO-E-2300 (Electronic Forms Systems)	√			
Data dictionary/directory services					
Data dictionary	NIST FIPS PUB 156 (IRDS)	√			

Table 7. Data Management Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: DATA MANAGEMENT SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Transaction processing					
Protocol for heterogeneous interoperability	ISO 10026-1,2,3:1992 (OSI Distributed Transaction Processing)	√			
Transaction manager-resource manager interface	X/Open C193:1992 (XA Specification)	√			
Transaction demarcation	X/Open P209:1992 (TX Specification)	√			
Transaction manager to communications manager interface (Complementary)	X/Open S423:1994 (XA+ Specification)	√			
	X/Open P306:1993 (XATMI Specification)	√			
	X/Open P305:1993 (TxRPC Specification)	√			
Distributed queuing	IEEE P1003.15 (POSIX Batch Extensions)	√			

Table 8 – Graphics Services Relationship to TENA

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: GRAPHICS SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Raster graphics					
Raster data Interchange (Alternative)	MIL-PRF-28002 (CALS Raster)	√			
	NIST FIPS PUB 150 (Group 4 Facsimile)	√			
	NIST FIPS PUB 158-1 (X-Windows, for BDF)	√			
Still image compression (Alternative)	NIST FIPS PUB 147 (Group 3 Compression)	√			
	NIST FIPS PUB 148 (General Facsimile)	√			
	NIST FIPS PUB 150 (Group 4 Facsimile)	√			
	ITU-T T.4-1988 (Group 3 Compression)	√			
	ITU-T T.6-1988 (Group 4 Compression)	√			

Table 8. Graphics Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: GRAPHICS SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
	ITU-T T.81-1993 (JPEG)	√			
	MIL-STD-188-196 (NITFS Bi-Level)	√			
	MIL-STD-188-197A (NITFS ARIDPCM)	√			
	MIL-STD-188-198A (NITFS JPEG)	√			
	MIL-STD-188-199 (NITFS Vector Quantization)	√			
	ISO/IEC 10918-1 (JPEG)	√			
Vector graphics					
Vector graphics API (Complementary)	NIST FIPS PUB 153 (PHIGS)	√			
	ISO/IEC 9592-4:1992 (PHIGS PLUS)	√			
Vector graphics data interchange (Alternative)	MIL-PRF-28000 (CALS IGES)	√			
	NIST FIPS PUB 177 (IGES)	√			
	MIL-PRF-28003 (CALS CGM)	√			
	MIL-STD-2301A (NITFS CGM)	√			
	NIST FIPS PUB 128-1 (CGM)	√			
Device interfaces					
Device interface API	ISO/IEC 9636-1..6:1991 (CGI)	√			

Table 9. Internationalization Services Relationship to TENA

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: INTERNATIONALIZATION SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Character set and data representation					
Coded character sets	ISO 6937:1994 (Coded Character Sets for Text Communication)	√			
7-Bit coded character sets (Complementary)	NIST FIPS PUB 1-2 (Code for Information Interchange)	√			
	ISO 646:1991 (ISO 7-Bit Coded Character Set for Information Exchange)	√			
8-Bit coded character sets	ISO 4873:1991 (ISO 8-Bit Code for Information Interchange)	√			
8-Bit single byte character sets	ISO 8859:1989 (ISO 8-Bit Single-Byte Coded Graphic Character Sets)	√			

Table 9. Internationalization Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: INTERNATIONALIZATION SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Control functions	ISO 6429:1992 (Control Functions for ISO 7-Bit and 8-bit Coded Character Sets)	√			
Code extension techniques	ISO 2022:1986 (ISO 7-Bit and 8-Bit Coded Character Sets - Code Extension Techniques)	√			
Universal character sets	ISO 10646-1:1993 (Universal Multiple-Octet Coded Character Set)	√			
Currency and funds representation	ISO 4217:1990 (Codes for the Representation of Currencies and Funds)	√			
Date and time representation (Complementary)	NIST FIPS PUB 4-1 (Representation of Calendar Date and Ordinal Date)	√			
	NIST FIPS PUB 58-1 (Representation of Local Time of Day)	√			
	NIST FIPS PUB 59 (Representations of Universal Time, Local Time Differentials, and US Time Zone References)	√			
Country name representation	TBD				
Representation of human sexes	TBD				
Representation of names of languages	TBD				
Cultural convention services					
Numerical value representation	TBD				
Customization to local norms (Complementary)	X/Open G304 (Internationalisation Guide, Version 2)	√			
	DOD HCI Style Guide	√			
Natural language support services					
Keyboard device layout	ISO 9995-1..8:1994 (Keyboard Device Layout)	√			
Related standards and programs					
Character set registration	TBD				

Table 10. Operating System Services Relationship to TENA

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: OPERATING SYSTEM SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Kernel operations					
File management services (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Input/output control (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
System operator services (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Process management and core operating system services (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Environment services (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Hardware error and event conditions (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
System resource limits	NIST SP 500-224 (OIW SIAs for OSEs)	√			
Message queues	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Login services	X/Open C434, C435, C436 (Single UNIX Specification)	√			
Storage device management	OSF DCE 1.1: DFS	√			
Threads interface (Alternative)	OSF DCE 1.1: Threads	√			
	IEEE 1003.1c (POSIX Threads Extension)	√			
Threads extension language binding	NIST SP 500-224 (OIW SIAs for OSEs)	√			
Kernal language bindings (Alternatives complementary to FIPS 151-2)	IEEE 1003.1b:1993, 1003.1g	√			
	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.5-1992 (POSIX Ada Language Interfaces)	√			

Table 10. Operating System Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: OPERATING SYSTEM SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
	IEEE 1003.9 (POSIX FORTRAN Binding)	√			
Media handling					
Backup and restore (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	NIST FIPS PUB 189 (POSIX.2)	√			
Floppy disk format and handling	NIST FIPS PUB 189 (POSIX.2)	√			
Data interchange format	MIL-STD-2045-14502-1A/2/3 (Internet Transport Profile)	√			
Network sequencing	MIL-STD-2045-14502-1A/2/3 (Internet Transport Profile)	√			
Shell and utilities					
Commands and utilities	NIST FIPS PUB 189 (POSIX.2)	√			
Print management (Alternative)	NIST FIPS PUB 189 (POSIX.2)	√			
	ISO 10175 (Document Printing Application)	√			
Language bindings to POSIX.2	NIST FIPS PUB 189 (POSIX.2)	√			
Shell programming language	NIST FIPS PUB 189 (POSIX.2)	√			
User-oriented commands and utilities	NIST FIPS PUB 189 (POSIX.2)	√			
File and program editing services	NIST FIPS PUB 189 (POSIX.2)	√			
Batch scheduling	NIST FIPS PUB 189 (POSIX.2)	√			
Real time extensions					
Memory management (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Scheduling (Complementary)	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
	NIST FIPS PUB 151-2 (POSIX.1)	√			
Semaphores	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Asynchronous I/O	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			

Table 10. Operating System Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: OPERATING SYSTEM SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Asynchronous event notification	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Synchronized I/O	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Real time file system	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
POSIX.1b language bindings	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Fault management services					
Fault management	NMF Omnipoint 1	√			
Clock/calendar services					
Clocks and timers	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Real time timers	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Distributed timing service	OSF DCE 1.1: DTS	√			
Operating system object services					
Object request broker	CORBA Specification Rev. 2.0, 1994		√		

Table 11. Software Engineering Services Relationship to TENA

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SOFTWARE ENGINEERING SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**√**	**Technical Architecture**	**TBD**
CASE tools and environments					
Software development environment	ANSI/IEEE 1209-1992 (Evaluation and Selection of CASE Tools)	√			
Specialized language and compiler tools (Alternative)	ISO/IEC 9945-2:1993 (POSIX, part 2: Shell and Utilities)	√			
	X/Open C436:1994 (Commands and Utilities)	√			

Table 11. Software Engineering Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SOFTWARE ENGINEERING SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
CASE tools and environments					
Software development environment	ANSI/IEEE 1209-1992 (Evaluation and Selection of CASE Tools)	√			
Specialized language and compiler tools (Alternative)	ISO/IEC 9945-2:1993 (POSIX, part 2: Shell and Utilities)	√			
	X/Open C436:1994 (Commands and Utilities)	√			
Software life cycle processes	[Pending completion of IEEE 1498/EIA 640, MIL-STD-498 is recommended for use subject to Agency/Service policy. ISO/IEC DIS 12207 Software Life Cycle Processes is currently in the international standardization process.] [In light of DoD's new policy on MIL-STDs, MIL-STD-498 is in the process of becoming an IEEE standard.]	√			
Software life cycle processes	MIL-STD-498 (Software Development and Documentation)	√			
Configuration management (Complementary)	ANSI/IEEE 828-1990 (Software Configuration Management Plans)	√			
	ANSI/IEEE 1042-1987 (Guide to Software Configuration Management)	√			
	MIL-STD-498 (Software Development and Documentation)	√			
Documentation	MIL-STD-498 (Software Development and Documentation)	√			
Joint reviews (Complementary)	ANSI/IEEE 1028-1988 (Software Reviews and Audits)			√	
	MIL-STD-498 (Software Development and Documentation)			√	
Software requirements (Complementary)	ANSI/IEEE 830-1984 (Guide to Software Requirements Specifications)	√			
	MIL-STD-498 (Software Development and Documentation)	√			
Software design Complementary)	ANSI/IEEE 1016-1987 (Recommended Practice for Software Design Descriptions)	√			
	ANSI/IEEE 1016.1-1993 (Guide for Software Design Descriptions)	√			
	ANSI/IEEE 990-1987 (Recommended Practices for Ada as a Program Design Language)	√			

Table 11. Software Engineering Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SOFTWARE ENGINEERING SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
	MIL-STD-498 (Software Development and Documentation)	√			
Software management indicators (Complementary)	MIL-STD-498 (Software Development and Documentation)	√			
	ISO/IEC 9126 (Quality Characteristics and Guidelines for their Use)	√			
	ANSI/IEEE 982.1-1988 (Standard Dictionary of Measures to Produce Reliable Software)	√			
	ANSI/IEEE 982.2-1988 (Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software)	√			
	ANSI/IEEE 1045-1992 (Software Productivity Metrics)	√			
	ANSI/IEEE 1061-1992 (Software Quality Metrics Methodology)	√			
Software testing and product evaluation (Complementary)	ANSI/IEEE 829-1983/R1991 (Software Test Documentation)	√			
	ANSI/IEEE 1008-1987 (Software Unit Testing)	√			
	NIST FIPS PUB 132 (Guide for Software Verification and Validation Plans)	√			
	ANSI/IEEE 1012-1987 (Software Verification and Validation Plans)	√			
	ANSI/IEEE 1059-1993 (Guide for Software Verification and Validation Plans)	√			
	MIL-STD-498 (Software Development and Documentation)	√			
Software quality assurance (Complementary - by sponsor)	ISO 9001:1987 (Model for Quality Assurance)	√			
	ISO 9000-3:1991 (Guidelines for Application of ISO 9001)	√			
	ANSI/IEEE 730.1-1989 (Software Quality Assurance Plans)	√			
	IEEE 1298-1992 (Software Quality Management System)	√			
	MIL-STD-498 (Software Development and Documentation)	√			

Table 11. Software Engineering Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SOFTWARE ENGINEERING SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Software problem categories/ priorities (Complementary)	IEEE 1044-1993 (Classification for Software Anomalies)	√			
	MIL-STD-498 (Software Development and Documentation)	√			
Software safety	MIL-STD-882 (System Safety Program Requirements)	√			
Software support (Complementary)	MIL-STD-498 (Software Development and Documentation)	√			
	ANSI/IEEE 1219-1993 (Software Maintenance)	√			
Software distribution	OSF DME: Distributed Services			√	
License management	OSF DME: License Management			√	
Languages					
Ada (Complementary)	ISO/IEC 8652:1995 (Ada95)	√			
	NIST FIPS PUB 119-1 (Ada95)	√			
C (Complementary)	ANSI/ISO 9899: (C)	√			
	NIST FIPS PUB 160	√			
FORTRAN (Alternative)	NIST FIPS PUB 69-1 (FORTRAN-77)	√			
	ISO 1539:1990 (FORTRAN-90)	√			
COBOL	NIST FIPS PUB 21-4 (COBOL)	√			
J OVIAL	MIL-STD-1589C, Notice 1, 1994 (JOVIAL)	√			
MUMPS (aka M)	NIST FIPS PUB 125-1 (MUMPS aka M)	√			
Bindings					
Ada bindings (Complementary)	ISO 9075:1992 (Binding to SQL)	√			
	ISO/ANSI 9593-3:1990 (Binding to PHIGS)	√			
	IEEE 1003.5-1992 (POSIX Ada Language Interfaces)	√			
	IEEE 1003.5b (POSIX Ada Real Time Binding)	√			
	ANSI X3.168-1989 (Embedded SQL and SQL Ada Module Extensions)	√			
	NIST FIPS PUB 127-2 (SQL, for Ada bindings)	√			

Table 12 – System Management Services Relationship to TENA

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SYSTEM MANAGEMENT SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
State management					
Independent window management services	OSF Motif AES 1.2	√			
Batch scheduling	NIST FIPS PUB 189 (POSIX.2)	√			
Process management and core operating system services (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
System administration and management APIs (Alternative)	NIST SP 500-224 (OIW SIAs for OSEs)	√			
	NMF Omnipoint 1	√			
	IEEE 1224	√			
	X/Open C206 (XMP)	√			
Scheduling (Complementary)	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)		√		
	NIST FIPS PUB 151-2 (POSIX.1)		√		
User/Group management					
User/Group identification (Complementary)	IEEE P1387.3		√		
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)		√		
Configuration control					
Software configuration management (Complementary)	ANSI/IEEE 828-1990 (Software Configuration Management Plans)	√			
	ANSI/IEEE 1042-1987 (Guide to Software Configuration Management)	√			
	MIL-STD-498 (Software Development and Documentation)	√			
Data dictionary	NIST FIPS PUB 156 (IRDS)	√			
System configuration	NMF Omnipoint 1	√			
Network configuration management	NMF Omnipoint 1	√			
Usage management and cost allocation					
Accounting management	NIST FIPS PUB 96	√			

Table 13. System Management Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SYSTEM MANAGEMENT SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Performance management					
Software management indicators (Complementary)	MIL-STD-498 (Software Development and Documentation)	√			
	ISO/IEC 9126 (Quality Characteristics and Guidelines for their Use)	√			
	ANSI/IEEE 982.1-1988 (Standard Dictionary of Measures to Produce Reliable Software)	√			
	ANSI/IEEE 982.2-1988 (Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software)	√			
	ANSI/IEEE 1045-1992 (Software Productivity Metrics)	√			
	ANSI/IEEE 1061-1992 (Software Quality Metrics Methodology)	√			
Performance management (Complementary)	NIST FIPS PUB 144	√			
	NMF Omnipoint 1	√			
Network flow control	MIL-STD-2045-14502-1A/2/3 (Internet Transport Profile)	√			
Network sequencing	MIL-STD-2045-14502-1A/2/3 (Internet Transport Profile)	√			
Communication of management information	MIL-STD-2045-38000	√			
Input/output control (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Event management (Alternative)	NMF Omnipoint 1	√			
	NIST SP 500-224 (OIW SIAs for OSEs)	√			
Fault management					
Software safety	MIL-STD-882 (System Safety Program Requirements)	√			
Network error recovery	MIL-STD-2045-14502-1A/2/3 (Internet Transport Profile)	√			
Fault management	NMF Omnipoint 1	√			
Storage device management	OSF DCE 1.1: DFS	√			

Table 13. System Management Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SYSTEM MANAGEMENT SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Backup and restore (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	NIST FIPS PUB 189 (POSIX.2)	√			
Hardware error and event conditions (Complementary)	NIST FIPS PUB 151-2 (POSIX.1)	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
Error and event logging	NMF Omnipoint 1	√			
Other management services					
Database administration	DoDD 8320.1 (DoD Data Administration)	√			
Floppy disk format and handling	NIST FIPS PUB 189 (POSIX.2)	√			
Print management (Complementary)	NIST FIPS PUB 189 (POSIX.2)	√			
	ISO 10175 (Document Printing Application)	√			

Table 14 – Security Services Relationship to TENA

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SECURITY SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Architectures and applications					
System development security (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DoD NCSC-TG-005, v1 (TNI)	√			
	DoD NCSC-TG-006, v1 (CM in Trusted Systems)	√			
	DoD NCSC-TG-021, v1 (TDI)	√			
	OSF DCE 1.1: Security	√			
	NIST FIPS PUB 151-2 (POSIX.1)	√			
	MIL-STD-498 (Software Development and Documentation)				

Table 14 – Security Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SECURITY SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Database security	NIST FIPS PUB 127-2:1993 (SQL)	√			
	NIST FIPS PUB 156 (IRDS)	√			
Network security architecture (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DoD NCSC-TG-005, v1 (TNI)	√			
	ISO 10181-2:1993 (OSI Authentication Framework)	√			
	NIST SP 500-224, pt 12,13 (OIW SIAs for OSEs)	√			
	ISO 10745:1993 (OSI Upper Layer Security Model)	√			
	ISO 11586-1:1994 (GULS, part 1)	√			
Operating system Security (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DDS-2600-5502-87 (CMW Security Requirements)	√			
	DDS-2600-6243-92 (CMW Evaluation Criteria)	√			
	DDS-2600-6216-91 (CMW Labeling Encoding Format)	√			
	DDS-2600-6243-91 (CMW Labeling Guidelines)	√			
	NIST FIPS PUB 151-2 (POSIX.1)	√			
	NIST FIPS PUB 112 (Password Usage)	√			
System management security					
Privacy act (Complementary)	PL 100-235 (Computer Security Act of 1987)			√	
	PL 93-579 (Privacy Act of 1974)			√	
Certification and accreditation	DoD 5200.28-STD (TCSEC)			√	
Security risk management (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	NIST FIPS PUB 191 (Guideline for LAN Security)	√			
Security Management (Complementary)	ISO 9595, AM4 (CMIS Access Control)	√			
	ISO 10164-7 (System Management Security Alarm Reporting)	√			
	ISO 10164-8 (System Management Security Audit Trail Function)	√			
	ITU-T X.518 (OSI Directory-Distributed Operations)	√			
	DoD 5200.28-STD (TCSEC)	√			
	ISO 9596-1 (CMIP)	√			
	DoD NCSC-TG-005, v1 (TNI)	√			

Table 14 – Security Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SECURITY SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
	DoD NCSC-TG-021, v1 (TDI)	√			
	NMF Omnipoint 1	√			
	IEEE 1003.1b:1993 (POSIX Real-Time Extensions)	√			
	NIST FIPS PUB 151-2 (POSIX.1)	√			
Security association and key management (Complementary)	NIUF ISDN Security Protocol 421 (SAMP)	√			
	ISO 11586-1:1994 (GULS, part 1)	√			
	ISO 11586-2 (GULS, part 2)	√			
	ISO 11586-3 (GULS, part 3)	√			
	NIST FIPS PUB 171 (Key Management Using ANSI X9.17)	√			
Security audit (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DoD NCSC-TG-005, v1 (TNI)	√			
	NMF Omnipoint 1	√			
	ISO 10164-8 (System Management Security Audit Trail Function)	√			
Security alarm reporting (Complementary)	ISO 10164-7 (System Management Security Alarm Reporting)				
	NMF Omnipoint 1				
Authentication					
Personal authentication (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	NIST FIPS PUB 112 (Password Usage)	√			
	NIST FIPS PUB 48 (Automated Personal ID)	√			
	ISO 9594-8.2 (OSI Directory Authentication Framework)	√			
Network authentication (Complementary)	MIL-STD-2045-18500 (MHS Message Security Protocol (MSP) Profile)	√			
	ITU-T X.509 (OSI Directory Authentication Framework)	√			
	DoD NCSC-TG-005, v1 (TNI)	√			
	NIST FIPS PUB 186 (DSS)	√			
	NIST FIPS PUB 180-1 (SHS)	√			
	ISO 8649 (OSI Service Definition for ACSE)	√			
	ISO 8650 (OSI Protocol Specification for ACSE)	√			
	ISO 11586-1:1994 (GULS, part 1)	√			
	ISO 11586-2 (GULS, part 2)	√			
	ISO 11586-3 (GULS, part 3)	√			
	ISO 11586-4 (GULS, part 4)	√			

Table 14 – Security Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SECURITY SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
	IEEE 802.10B-1992 (SILS Secure Data Exchange)	√			
Entity authentication (Complementary)	NIST FIPS PUB 113 (Computer Data Authentication)	√			
	DoD 5200.28-STD (TCSEC)	√			
	ISO 9807 (Retail Message Authentication)	√			
	ISO 9798-1 (Entity Authentication Mechanism)	√			
	ISO 9798-3 (Entity Authentication Mechanism)	√			
Access control					
System access control (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	ISO 9595, AM4 (CMIS Access Control)	√			
Network access control (Complementary)	ISO 9595, AM4 (CMIS Access Control)	√			
	MIL-STD-2045-18500 (MHS Message Security Protocol (MSP) Profile)	√			
	DoD NCSC-TG-005, v1 (TNI)	√			
	IEEE 802.10B-1992 (SILS Secure Data Exchange)	√			
Confidentiality					
Open systems confidentiality (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	PL 93-579 (Privacy Act of 1974)		√		
	PL 100-235 (Computer Security Act of 1987)		√		
Data encryption security (Complementary)	NIST FIPS PUB 46-2 (DES)	√			
	NIST FIPS PUB 74 (Guidelines for DES)	√			
	NIST FIPS PUB 81 (DES Modes of Operation)	√			
	NIST FIPS PUB 185 (EES)	√			
	NIST FIPS PUB 140-1 (Security Requirements for Cryptographic Modules)	√			
	ISO 8372 (Modes of Operation for a 64-Bit Block Cipher Algorithm)	√			
Traffic flow confidentiality	ISO 11577:1994 (NLSP)	√			

Table 14 – Security Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SECURITY SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Integrity					
Open systems	DoD 5200.28-STD (TCSEC)	√			
integrity (Complementary)	DoD NCSC-TG-021, v1 (TDI)	√			
Data integrity techniques (Complementary)	NIST FIPS PUB 46-2 (DES)	√			
	NIST FIPS PUB 74 (Guidelines for DES)	√			
	NIST FIPS PUB 81 (DES Modes of Operation)	√			
	NIST FIPS PUB 185 (EES)	√			
	NIST FIPS PUB 140-1 (Security Requirements for Cryptographic Modules)	√			
	ISO 8372 (Modes of Operation for a 64-Bit Block Cipher Algorithm)	√			
	NIST FIPS PUB 180-1 (SHS)	√			
	NIST FIPS PUB 186 (DSS)	√			
Network integrity Complementary)	ISO 11586-1:1994 (GULS, part 1)	√			
	ISO 11586-4 (GULS, part 4)	√			
	IEEE 802.10B-1992 (SILS Secure Data Exchange)	√			
	ITU-T X.500:1993 (OSI Directory (ISO 9594)	√			
Non-repudiation					
Open systems non-repudiation (Complementary)	MIL-STD-2045-18500 (MHS Message Security Protocol (MSP) Profile)	√			
	NIST FIPS PUB 186 (DSS)	√			
	ISO 11586-1:1994 (GULS, part 1)	√			
	ISO 11586-4 (GULS, part 4)	√			
Electronic signature	NIST FIPS PUB 186 (DSS)	√			
Electronic hashing	NIST FIPS PUB 180-1 (SHS)	√			
Availability					
Detection and notification (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DoD NCSC-TG-005, v1 (TNI)	√			
Security recovery (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DoD NCSC-TG-005, v1 (TNI)	√			

Table 14 – Security Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SECURITY SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Security labeling					
User interface security labeling (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DoD HCI Style Guide, v. 3.0; TAFIM Vol. 8	√			
	DDS-2600-6243-92 (CMW Evaluation Criteria)	√			
	DDS-2600-6243-91 (CMW Labeling Guidelines)	√			
	DDS-2600-6216-91 (CMW Labeling Encoding Format)	√			
	DoDIIS Style Guide	√			
Data management security labeling (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DDS-2600-6243-92 (CMW Evaluation Criteria)	√			
	DDS-2600-6243-91 (CMW Labeling Guidelines)	√			
	DDS-2600-6216-91 (CMW Labeling Encoding Format)	√			
Data interchange security labeling Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DDS-2600-6243-92 (CMW Evaluation Criteria)	√			
	DDS-2600-6243-91 (CMW Labeling Guidelines)	√			
	DDS-2600-6216-91 (CMW Labeling Encoding Format)	√			
	MIL-STD-2045-48501 (Common Security Label (CSL)	√			
	ITU-T X.411 (MHS Message Transfer System: Abstract Service Definition and Procedures)	√			
Graphics security labeling (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DDS-2600-6243-92 (CMW Evaluation Criteria)	√			
	DDS-2600-6243-91 (CMW Labeling Guidelines)	√			
	DDS-2600-6216-91 (CMW Labeling Encoding Format)	√			
Data communications security labeling	MIL-STD-2045-48501 (Common Security Label (CSL))	√			
	DoD 5200.28-STD (TCSEC)	√			

Table 14 – Security Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: SECURITY SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
(Complementary)	DDS-2600-6243-92 (CMW Evaluation Criteria)	√			
	DDS-2600-6243-91 (CMW Labeling Guidelines)	√			
	DDS-2600-6216-91 (CMW Labeling Encoding Format)	√			
Operating system security labeling (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DDS-2600-6243-92 (CMW Evaluation Criteria)	√			
	DDS-2600-6243-91 (CMW Labeling Guidelines)	√			
	DDS-2600-6216-91 (CMW Labeling Encoding Format)	√			
Distributed computing security labeling (Complementary)	DoD 5200.28-STD (TCSEC)	√			
	DoD NCSC-TG-005, v1 (TNI)	√			
	DoD NCSC-TG-021, v1 (TDI)	√			
	DDS-2600-6243-92 (CMW Evaluation Criteria)	√			
	DDS-2600-6243-91 (CMW Labeling Guidelines)	√			
	DDS-2600-6216-91 (CMW Labeling Encoding Format)	√			

Table 15 – User Interface Services Relationship to TENA

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: USER INTERFACE SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
User Interface					
Keyboard device layout	ISO 9995-1..8:1994 (Keyboard Device Layout)	√			
Graphical Client-Server Operations					
Data stream encoding	NIST FIPS PUB 158-1 (X-Windows)	√			
Data stream interface	NIST FIPS PUB 158-1 (X-Windows)	√			
Subroutine foundation library	NIST FIPS PUB 158-1 (X-Windows)	√			

Table 15 – User Interface Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: USER INTERFACE SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Raster data interchange (Alternative)	MIL-PRF-28002 (CALS Raster)	√			
	NIST FIPS PUB 150 (Group 4 Facsimile)	√			
	NIST FIPS PUB 158-1 (X-Windows, for BDF)	√			
User interface management system	NIST FIPS PUB 158-1 (X-Windows)	√			
Communication between GUI client applications	OSF Motif AES 1.2: ICCCM, v 1.0	√			
Data interchange format for GUI-based applications (Complementary)	OSF Motif AES 1.2: ICCCM, v 1.0	√			
	NIST FIPS PUB 158-1 (X-Windows)	√			
Compound text encoding	X/Open CTE, v1.1	√			
X logical font description	X/Open XLFD, v1.3	√			
Object definition and management					
3-D appearance	NIST FIPS PUB 158-1 (X-Windows, for PEX)	√			
GUI internationali-zation support	X/Open G304:1993 (Internationalisation Guide)	√			
Interchange format for design tools	COSE Motif	√			
Application programming interfaces	IEEE 1295-1993 (Motif)	√			
Language bindings for bit-mapped GUIs	IEEE 1295-1993 (Motif)	√			
Style guide	DoD HCI Style Guide, v. 3.0; TAFIM Vol. 8	√			
User interface definition language	OSF Motif AES 1.2: UIDL	√			

Table 15 – User Interface Services Relationship to TENA (Cont'd)

TAFIM Categorization		TENA Relationship			
MAJOR SERVICE AREA: USER INTERFACE SERVICES					
Mid and Base Service Areas (Indented)	**Adopted Standard or Specification**	**Facility**	**System Architecture**	**Technical Architecture**	**TBD**
Window management					
Independent window management services	OSF Motif AES 1.2	√			
Multiple displays	OSF Motif AES 1.2	√			
Style guide	DoD HCI Style Guide, v. 3.0; TAFIM Vol. 8	√			
Drivability	DoD HCI Style Guide, v. 3.0; TAFIM Vol. 8	√			
On-line help	DoD HCI Style Guide, v. 3.0; TAFIM Vol. 8	√			
Commands, menus, and dialog	DoD HCI Style Guide, v. 3.0; TAFIM Vol. 8	√			
Character-based user interface					
Style guide	DoD HCI Style Guide, v. 3.0; TAFIM Vol. 8	√			
Electronic forms	JIEO-E-2300 (Electronic Forms Systems)	√			

Table 16. JTA Standards

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	**System Architecture**	**Technical Architecture**	**TBD**
2.2.2.1.2 USER INTERFACE SERVICES				
Win32 APIs, Window Management and Graphics Device Interface, Volume 1 Microsoft Win32 Programmers Reference Manual, 1993, Microsoft Press	√			
X/Open C323, Common Desktop Environment (CDE) Version 1.0, April 1995 production of documents which are intended for long-term storage and electronic dissemination for viewing in multiple formats.	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
2.2.2.1.3 DATA MANAGEMENT SERVICES	√			
Open Data Base Connectivity, ODBC 2.0	√			
2.2.2.1.4 DATA INTERCHANGE SERVICES				
See Section 4.x				
2.2.2.1.4.1 Document interchange				
ISO 8879: 1986, Standard Generalized Markup Language (SGML), for the RFC-1866: 1995, Hypertext Mark-up Language (HTML), Internet Version 2.0	√			
See Table 2-1				
2.2.2.1.4.3 Geospatial Data Interchange				
MIL-STD-2407, Interface Standard for Vector Product Format (VPF)	√			
Defense Mapping Agency (DMA) DMA List of Products and Services	√			
2.2.2.1.4.4 Imagery Data Interchange				
ANSI/ISO 8632: 1992, Computer Graphics Metafile (CGM) as profiled by FIPS 128 and MIL-STD-2301	√			
2.2.2.1.4.6 Audio Data Interchange				
ISO/IEC 11172-1: 1993 - Encoding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s -- Part 1: Systems	√			
ISO/IEC 11172-3: 1993 - Encoding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s -- Part 3: Audio	√			
ISO/IEC 11172-3/Cor. 1: 1995 - Encoding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s -- Part 3: Audio Technical Corrigendum	√			
ISO 13818-1: 1996 - Generic Coding of Moving Pictures and Associated Audio Information - Part 1: Systems	√			
ISO 13818-3: 1995 - Generic Coding of Moving Pictures and Associated Audio Information - Part 3: Audio	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
2.2.2.1.4.7 Video Data Interchange				
ISO/IEC 11172-1: 1993 - Encoding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s -- Part 1: Systems	√			
ISO/IEC 11172-1: 1993/Cor. 1:1995 Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s -- Part 1: Systems Technical Corrigendum 1	√			
ISO/IEC 11172-2: 1993 Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s -- Part 2 Video	√			
ISO 13818-1: 1996 - Generic Coding of Moving Pictures and Associated Audio Information - Part 1: Systems	√			
ISO 13818-2: 1996 - Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video	√			
2.2.2.1.4.8 Atmospheric Data Interchange				
FM 92-X-GRIB - The WMO Format for the Storage of Weather Product Information and the Exchange of Weather Product Messages in Gridded Binary (GRIB) Form	√			
FM 94-X-BUFR - The WMO Binary Universal Format for Representation (BUFR) of meteorological data	√			
Data Exchange Format (DEF) - Appendix 30 to the Tactical Automated Weather Distribution System (TAWDS)/Integrated Meteorological System (IMETS) Implementation Document for Communication Information Data Exchange (CIDE)	√			
2.2.2.1.4.8 Oceanographic Data Interchange				
FM 94-X-BUFR - The WMO Binary Universal Format for Representation (BUFR) of oceanographic data	√			
2.2.2.1.5 Graphics Services				
ISO 7942 as profiled by FIPS Pub 120-1 (change notice 1): 1991, Graphical Kernel System (GKS) - for 2-D graphics	√			
ISO 9592: 1989, as profiled by FIPS Pub 153, Programmers Hierarchical Interactive Graphics Systems (PHIGS) - for 3-D graphics	√			
ISO/IEC 9636: 1994, Information Technology-Computer Graphics-Interfacing (CGI) Techniques for Dialogue with Graphics Devices	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
2.2.2.1.7 Operating System Services				
ISO 9945-1: 1990, Information Technology - Portable Operating System Interface for Computer Environments (POSIX) - Part 1: System Application Program Interface (API) [C language]	√			
ISO 9945-2: 1993, Information Technology - Portable Operating System Interface for Computer Environments (POSIX) - Part 2: Shell and Utilities	√			
IEEE 1003.2d: 1994, POSIX - Part 2: Shell and Utilities - Amendment: Batch Environment	√			
IEEE 1003.1i: 1995, POSIX - Part 1: System Application Program Interface (API) Amendment: Technical Corrigenda to Real-time Extension [C Language]	√			
Win32 APIs, Window Management and Graphics Device Interface, Volume 1 Microsoft Win32 Programmers Reference Manual, 1993	√			
2.2.2.2.1 Internationalization Services				
ISO/IEC 8859-1: 1987, Information Processing - 8-Bit Single-Byte Coded Character Sets - Part 1: Latin Alphabet No. 1	√			
ISO/IEC 10646-1: 1993, Information Technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane	√			
2.2.2.2.4.1 Remote Procedure Computing				
OSF - DCE Remote Procedure Call (RPC), Version 1.1, 1994	√			
OSF - DCE Time Services, Version 1.1, 1994	√			
OSF - DCE Directory Services, Version 1.1, 1994	√			
2.2.2.2.4.2 Distributed Object Computing				
OMG - The Common Object Request Broker: Architecture and Specification, Version 2: July 1995	√			
OMG - CORBA services: Common Object Services Specification, March 1996	√			
OMG - CORBA facilities: Common Object Facilities Architecture, November 1995	√			
2.3.3 Data Management				
SQL3	√			
ODMG-9x standard	√			
ISO 9075-3, 1995 Call Level Interface	√			
DIS 9075-4, Database Language, Part 4: Persistent Stored Modules (SQL/PSM)	√			
Open Data Base Connectivity (ODBC) 3.0	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
2.3.4 Data Interchange				
ISO 13818-4, MPEG-2 is an interchange format used for full motion video and associated audio data for data rates of 1.5 Mbits/s - 6.0 Mbits/s	√			
HTML 3.2	√			
MIL-STD-2405, Datums, Coordinates and Grids	√			
MIL-STD-600001, Accurace	√			
2.3.5 Data Interchange				
P1003.1d - Real-Time Extensions	√			
P1003.1h - Services for Reliable, Available, Serviceable Systems	√			
P1003.5b - Ada Bindings for Real-Time	√			
P1003.2l - Real-Time Distributed Systems Communication	√			
P1003.1j - Advanced Real-Time Extensions	√			
3.2.1.1 Host Standards				
IAB Standard 3/RFC-1122/RFC-1123, Host Requirements, October 1989	√			
3.2.1.1.1 Application Support Standards				
3.2.1.1.1.1 Electronic Mail				
ACP 123 U.S. Supplement No. 1, Common Messaging Strategy and Procedures, November 1995	√			
3.2.1.1.1.2 Directory Services				
3.2.1.1.1.2.2 Domain Name System (DNS)				
IAB Standard 13/RFC-1034/RFC-1035, Domain Name System, November 1987	√			
3.2.1.1.1.3 File Transfer				
IAB Standard 9/RFC-959, File Transfer Protocol, October 1985, with the following FTP commands mandated for reception: Store unique (STOU) and Abort (ABOR)	√			
3.2.1.1.1.4 Remote Terminal				
IAB Standard 8/RFC-854/RFC-855, TELNET Protocol, May 1983	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
3.2.1.1.1.5 Network Management				
IAB Standard 15/RFC-1157, Simple Network Management Protocol (SNMP), May 1990	√			
IAB Standard 16/RFC-1155/RFC-1212, Structure of Management Information, May 1990	√			
IAB Standard 17/RFC-1213, Management Information Base, March 1991	√			
3.2.1.1.1.6 Network Time				
RFC-1305, Network Time Protocol (V3), April 9, 1992	√			
3.2.1.1.1.7 Bootstrap Protocol (BOOTP)				
RFC-951, Bootstrap Protocol, September 1, 1985	√			
RFC-1533, DHCP Options and BOOTP Vendor Extensions, October 8, 1993	√			
RFC-1542, Clarifications and Extensions for the Bootstrap Protocol, October 27, 1993	√			
3.2.1.1.1.8 Dynamic Host Configuration Protocol (DHCP)				
RFC-1541, Dynamic Host Configuration Protocol, October 27, 1993	√			
3.2.1.1.1.9 World Wide Web (WWW) Services				
3.2.1.1.1.9.1 Hypertext Transfer Protocol (HTTP)				
RFC-1945, Hypertext Transfer Protocol -- HTTP/1.0, May 17, 1996	√			
3.2.1.1.1.9.2 Uniform Resource Locator (URL)				
RFC-1738, Uniform Resource Locators, December 20, 1994	√			
RFC-1808, Relative Uniform Resource Locators, June 14, 1995	√			
3.2.1.1.1.10 Connectionless Data Transfer				
MIL-STD-2045-47001, Connectionless Data Transfer Application Layer Standard, July 27, 1995	√			
3.2.1.1.2.1 Transmission Control Protocol (TCP)/User Datagram Protocol (UDP) over Internet Protocol (IP)				
3.2.1.1.2.1.1 Transmission Control Protocol (TCP)				
addition, TCP shall implement the PUSH flag and the Nagle Algorithm, as defined in IAB Standard 3	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
3.2.1.1.2.1.2 User Datagram Protocol (UDP)				
IAB Standard 6/RFC-768, User Datagram Protocol, August 1980	√			
3.2.1.1.2.1.3 Internet Protocol (IP)				
IAB Standard 5/RFC-791/RFC-950/RFC-919/RFC-922/RFC-792/RFC-1112, Internet Protocol, September 1981	√			
3.2.1.1.2.2 Open Systems Interconnection (OSI)/Internet Interworking Protocol				
IAB Standard 35/RFC 1006, ISO Transport Service on top of the TCP, May 1978	√			
3.2.1.2 Video Teleconferencing (VTC) Standards				
VTC001, Industry Profile for Video Teleconferencing, Revision 1, April 25, 1995	√			
ITU-T H.324, Terminal for Low Bit Rate Multimedia Communications, March 19, 1996	√			
3.2.1.3 Facsimile Standards				
3.2.1.3.1 Analog Facsimile Standard				
TIA/EIA-465-A, Group 3 Facsimile Apparatus for Document Transmission, March 21, 1995	√			
TIA/EIA-466, Procedures for Document Facsimile Transmission, May 1981	√			
3.2.1.3.2 Digital Facsimile Standard				
MIL-STD 188-161D, Interoperability and Performance Standards for Digital Facsimile Equipment, January 10, 1995	√			
3.2.2 Network Standards				
3.2.2.1 Router Standards				
RFC-1812, Requirements for IP Version 4 Routers, June 22, 1995	√			
IAB Standard 6/RFC-768, User Datagram Protocol, August 1980	√			
IAB Standard 7/RFC-793, Transmission Control Protocol, September 1981	√			
IAB Standard 8/RFC-854/RFC-855, TELNET Protocol, May 1983	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	**System Architecture**	**Technical Architecture**	**TBD**
IAB Standard 13/RFC-1034/RFC-1035, Domain Name System, November 1987	√			
IAB Standard 15/RFC-1157, Simple Network Management Protocol, May 1990	√			
IAB Standard 16/RFC-1155/RFC-1212, Structure of Management Information, May 1990	√			
IAB Standard 17/RFC-1213, Management Information Base, March 1991	√			
RFC-951, Bootstrap Protocol, September 1, 1985	√			
RFC-1533, DHCP Options and BOOTP Vendor Extensions, October 8, 1993	√			
RFC-1541, DHCP, October 27, 1993	√			
RFC-1542, Clarifications and Extensions for the Bootstrap Protocol, October 27, 1993	√			
IAB Standard 33/RFC-1350, Trivial FTP (TFTP), July 1992, to be used for initialization only	√			
3.2.2.1.1 Internet Protocol (IP)				
IAB Standard 5/RFC-791/RFC-950/RFC-919/RFC-922/RFC-792/RFC-1112, Internet Protocol, September 1981	√			
3.2.2.1.2 IP Routing				
3.2.2.1.2.1 Interior Routers				
RFC-1583, Open Shortest Path First Routing Version 2, March 23, 1994, for unicast routing	√			
RFC-1584, Multicast Extensions to OSPF, March 24, 1994, for multicast routing	√			
3.2.2.1.2.2. Exterior Routers				
RFC-1771, Border Gateway Protocol 4, March 21, 1995	√			
RFC-1772, Application of BGP-4 In the Internet, March 21, 1995	√			
	√			
3.2.2.2 Subnetworks				
3.2.2.2.1 Local Area Network (LAN) Access				
ISO/IEC 8802-3:1993, Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, 10BaseT Medium-Access Unit (MAU)	√			
IAB-Standard 41/RFC-894, Standard for the Transmission of IP Datagrams Over Ethernet Networks, April 1984	√			
IAB Standard 37/RFC-826, An Ethernet Address Resolution Protocol, November 1982	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
3.2.2.2.2 Point to Point Standards				
IAB Standard 51/RFC-1661/RFC-1662, Point-to-Point Protocol (PPP), July 1994	√			
RFC-1332, PPP Internet Protocol Control Protocol (IPCP), May 26, 1992	√			
RFC-1333, PPP Link Quality Monitoring, May 26, 1992	√			
RFC-1334, PPP Authentication Protocols, October 20, 1992	√			
RFC-1570, PPP Link Control Protocol (LCP) Extensions, January 11, 1994	√			
EIA 232E, Interface Between Data Terminal Equipment and Data Circuit Terminating Equipment Employing Serial Binary Data Interchange, July 1991	√			
EIA 449, General Purpose 37-Position and 9-Position Interface for Data Terminal Equipment and Data Circuit Terminating Equipment Employing Serial Binary Data Interchange, February 19	√			
EIA 530A, High Speed 25-Position Interface for Data Terminal Equipment and Data Circuit Terminating Equipment, June 1992, Including Alternate 26-Position Connector, 1992	√			
3.2.2.2.4 Integrated Services Digital Network (ISDN)				
ANSI T1.601, Telecommunications - Integrated Services Digital Network (ISDN) - Basic Access Interface for Use on Metallic loops for Application on the Network Side of the NT (Layer 1 Specification), 1992	√			
ANSI T1.408, Telecommunications - Integrated Services Digital Network (ISDN) - Primary Rate - Customer Installation Metallic Interfaces (Layer 1 Specification), 1990	√			
ITU-T Q.921, ISDN User-Network Interface - Data Link Layer Specification - Digital Subscriber Signaling System No. 1, 1993	√			
ITU-T Q.931, ISDN User-Network Interface Layer 3 Specification for basic Call Control - Digital Subscriber Signaling System No. 1(DSS 1), Network Layer, User-Network Management, 1989	√			
ITU-T E.164, Numbering Plan for the ISDN Era, 1991	√			
DCAC 370-175-13, Defense Switched Network System Interface Criteria, section titled Worldwide Numbering and Dialing Plan (WNDP), September 1993	√			
RFC-1356, Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode, August 6, 1992.	√			
For transmitting IP packets using Point-to-Point Protocol (PPP) over ISDN RFC-1618, PPP over ISDN, May 13, 1994	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
3.2.2.2.5 Asynchronous Transfer Mode (ATM)				
ATM Forum's UNI Specification V 3.1, User-Network Interface, September 1994	√			
ANSI T1.630 ATM Adaptation Layer for Constant Bit Rate Services Functionality and Specification, 1993	√			
ANSI T1.635 ATM Adaptation Layer Type 5 Common Part Functions and Specifications, 1994, which adopts ITU-T I.363, section 6	√			
RFC-1577, Classical IP and Address Resolution Protocol (ARP) over ATM, January 20, 1994	√			
3.2.3 Transmission Media				
3.2.3.1 Military Satellite Communications (MILSATCOM)				
3.2.3.1.1 Ultra High Frequency (UHF) Satellite Terminal Standards				
3.2.3.1.1.1 5- and 25-kHz Service				
MIL-STD-188-181, Interoperability Standard for Dedicated 5-kHz and 25-kHz UHF Satellite Communications, 18 September 1992	√			
3.2.3.1.1.2 5-kHz Demand Assigned Multiple Access (DAMA) Service				
MIL-STD-188-182, Interoperability Standard for 5 kHz UHF DAMA Terminal Waveform, 18 September 1992	√			
3.2.3.1.1.3 25-kHz Time Division Multiple Access (TDMA)/Demand Assigned Multiple Access (DAMA) Service				
MIL-STD-188-183, Interoperability Standard for 25 kHz UHF/TDMA/DAMA Terminal Waveform, 18 September 1992	√			
3.2.3.1.1.4 Data Control Waveform				
MIL-STD-188-184, Interoperability and Performance Standard for the Data Control Waveform, 20 August 1993	√			
3.2.3.1.2 Super High Frequency (SHF) Satellite Terminal Standards				

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
3.2.3.1.2.1 Earth Terminals				
MIL-STD-188-164, Interoperability and Performance Standards for C-Band, X-Band, and Ku-Band SHF Satellite Communications Earth Terminals, 13 January 1995	√			
3.2.3.1.2.2 Phase Shift Keying (PSK) Modems				
MIL-STD-188-165, Interoperability and Performance Standards for SHF Satellite Communications PSK Modems (Frequency Division Multiple Access (FDMA) Operations), January 13, 1995	√			
3.2.3.1.3 Extremely High Frequency (EHF) Satellite Payload and Terminal Standards				
3.2.3.1.3.1 Low Data Rate (LDR)				
MIL-STD-1582, EHF LDR Uplinks and Downlinks, December 10, 1992	√			
3.2.3.1.3.2 Medium Data Rate (MDR)				
MIL-STD-188-136, EHF MDR Uplinks and Downlinks, August 26, 1995	√			
3.2.3.2 Radio Communications				
3.2.3.2.1 High Frequency (HF)				
3.2.3.2.1.1 Automated Link Establishment (ALE)				
MIL-STD-188-141A, Medium and High Frequency Radio Equipment Standard, September 10, 1993	√			
3.2.3.2.1.2 Anti-jamming Capability				
MIL-STD-188-148, Interoperability Standard Anti-Jam Communications (2-30 Mhz), April 13, 1992	√			
3.2.3.2.1.3 Data Modems				
MIL-STD-188-110A, Data Modems, Interoperability and Performance Standards, September 30, 1991	√			
3.2.3.2.2 Very High Frequency (VHF)				
MIL-STD-188-242, Tactical Single Channel (VHF) Radio Equipment, June 20, 1985	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
3.2.3.2.3 Ultra High Frequency (UHF)				
MIL-STD-188-243, Tactical Single Channel (UHF) Radio Communications, March 15, 1989	√			
3.2.3.2.4 Super High Frequency (SHF)				
MIL-STD-188-145, Digital Line-of-Sight (LOS) Microwave Radio Equipment, July 28, 1992	√			
3.2.3.2.5 JTIDS/MIDS Transmission Media				
JTIDS System Segment Specification (Class 2 Terminal)-	√			
STANAG 4175, Edition 1, 29 August 1991 - Technical Characteristics of the Multifunctional Information Distribution System (MIDS)	√			
3.2.3.3 Synchronous Optical Network (SONET) Transmission Facilities				
ANSI T1.105, Telecommunications - Synchronous Optical Network (SONET) Basic Description Including Multiplex Structure, Rates and Formats (ATIS) (Revision and Consolidation of ANSI T1.105-1991 and ANSI T1.105A-1991), 1995	√			
ANSI T1.107 Digital Hierarchy - Formats Specifications, 1995	√			
ANSI T1.117, Digital Hierarchy - Optical Interface Specifications (SONET) (Single Mode - Short Reach), 1991	√			
3.3.2.1 Internet Standards				
RFC-1883, IPv6 Specification	√			
RFC-1884, IPv6 Addressing Architecture	√			
RFC-1885, ICMPv6 for IPv6	√			
RFC-1886, DNS Extensions to support IPv6	√			
3.3.2.2 Video Teleconferencing (VTC) Standards				
ITU H.321 VTC over ATM	√			
ITU H.323 VTC Ethernet networks	√			
3.3.2.3 Global Positioning System (GPS)				
ASD Command, Control, Communications, and Intelligence (C3I) Memorandum Development, Procurement, and Employment of DoD Global Position System, User Equipment, dated 31 April 1992	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
3.3.3 Network Standards				
3.3.3.1 Network Access Protocols				
IEEE 802.11 Wireless LAN	√			
IEEE 802.3u Fast Ethernet	√			
ATM LAN Emulation, Version 1.0	√			
IS-41 Public switched telephone networks (PSTN)	√			
IS-54 TDMA	√			
IS-95 Code Division Multiple Access (CDMA)	√			
Future Public Land Mobile Telecommunications Systems (FPLMTS) standards	√			
3.3.3.2 Link 22 Transmission Standards				
Link 22 Transmission media, standard is under development	√			
3.3.4 Military Satellite Communications (MILSATCOM)				
MIL-STD-188-166 (Interface Standard, Interoperability and Performance of Non-Electronic Protective Measures (EPM) for SHF SATCOM Link Control Protocols and Messaging Standards)	√			
MIL-STD-188-167 (Interface Standard, Message Format for SHF SATCOM Demand Assignment)	√			
MIL-STD-188-168 (Interface Standard, Interoperability and Performance Standards for SHF Satellite Communications Mulitplexers and Demultiplexers)	√			
MIL-STD-188-185 (Interface Standard, Interoperability of UHF MILSATCOM DAMA Control System).	√			
4.2 MANDATES				
4.2.1 Activity Model				
FIPS PUB 183, Integration Definition for Function Modeling (IDEF0).		√		
4.2.2 Data Model				
DoD Manual 8320.1-M-1, DoD Data Standardization Procedures	√			
FIPS PUB 184, Integration Definition For Information Modeling (IDEF1X). December 1993.		√		
4.2.3 DoD Data Definitions				
DoD Manual 8320.1-M-1, DoD Data Standardization Procedures	√			
Defense Data Dictionary System (DDDS).	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
4.2.4.2 Tactical Information Standards				
4.2.4.2.1 Bit-Oriented Data				
VMF Technical Interface Design Plan - Test Edition (TIDP-TE), Reissue 1 February 1995.	√			
4.2.4.2.2 US Message Text Format (USMTF) Messages				
MIL-STD-6040, United States Message Text Format (USMTF)	√			
4.2.4.2.3. Database-to-Database Exchange				
Database-to-Database Exchange shall use standard data elements from DDDS	√			
4.3 EMERGING STANDARDS				
4.3.2 Data Modeling				
IDEF1X97, Conceptual Schema Modeling (standard for data modeling)		√		
4.3.4 Information Standards				
Multi-functional Information Distribution System (MIDS)	√			
5.2.2 Style Guides				
5.2.2.1 Commercial Style Guides				
Open Software Foundation (OSF)/MotifÔ Style Guide, Revision 1.2 (OSF 1992)	√			
The WindowsÔ Interface: An Application Design Guide, Microsoft Press, 1992	√			
5.2.2.3 Domain-level Style Guides				
User Interface Specification for the Defense Information Infrastructure (DII), June 1996.	√			
5.3 EMERGING STANDARDS				
MIL-STD-2525A - This standard provides common warfighting symbology	√			
6.2.2 Information Processing Security Standards				
6.2.2.1 Application Software Entity Security Standards				
DoD 5200.28-STD, The DoD Trusted Computer System Evaluation Criteria, December 1985.	√			
NCSC-TG-021, Version 1, Trusted Database Management System Interpretation, April 1991.	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
FORTEZZA Application Implementors' Guide, MD4002101-1.52, 5 March 1996	√			
FORTEZZA Cryptologic InterfaceProgrammers' Guide, MD4000501-1.52, 30 January 1996.	√			
6.2.2.2.1 Data Management Services				
NCSC-TG-021, Version 1, Trusted Database Management System Interpretation, April 1991.	√			
6.2.2.2.2.2 Authentication Security Standards				
RFC-1510, The Kerberos Network Authentication Service, V.5, 10 September 1993	√			
6.2.3 Information Transfer Security Standards				
6.2.3.1 End System Security Standards				
6.2.3.1.1 Host Security Standards				
FORTEZZA Interface Control Document, Revision P1.5, 22 December 1994	√			
FORTEZZA Plus Interface Control Document, Release 3.0, 1 June 1995	√			
6.2.3.1.1.1 Security Algorithms				
Key Exchange Algorithm, NSA, R21-TECH-23-94, 12 July 1994	√			
6.2.3.1.1.2 Security Protocols				
SDN.903, revision 3.2, Secure Data Network System (SDNS) Key Management Protocol (KMP), August 1, 1989	√			
6.2.3.1.1.3 Evaluation Criteria Security Standards				
NCSC-TG-005, Version-1, Trusted Network Interpretation, July 1987	√			
6.2.3.2 Network Security Standards				
6.2.3.2.1 Internetworking Security Standards				
SDN.301, revision 1.5, Secure Data Network System (SDNS) Security Protocol 3 (SP3), 1989	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
6.3.2 Information Processing Security Standards				
6.3.2.1 Application Software Entity Security Standards				
6.3.2.1.1 Evaluation Criteria Security Standards				
Common Criteria for Information Technology Security Evaluation	√			
6.3.2.2 Application Platform Entity Security Standards				
6.3.2.2.1 Software Engineering Services Security				
6.3.2.2.1.1 Generic Security Service (GSS)-Application Program Interface (API) Security				
RFC-1508 defines GSS-API services and primitives at a level independent of underlying mechanism and programming language environment	√			
6.3.2.2.1.2 POSIX Security Standards				
IEEE P1003.1e, POSIX Part 1: System API - Protection, Audit, and Control Interfaces [C Language], Draft 15 (reballot March 1996)	√			
IEEE P1003.2c, POSIX Part 2: Shell and Utilities - Protection and Control Interfaces, Draft 15 (reballot March 1996)	√			
6.3.2.2.2 Authentication Security Standards				
RFC-1938, A One-Time Password System	√			
6.3.2.2.3 Distributed Computing Services Security Standards				
The Common Object Request Broker Architecture (CORBA), OMG 95-12-1, December 1995.	√			
6.3.3 Information Transfer Security Standards				
6.3.3.1 End System Security Standards				
6.3.3.1.1 Host Security Standards				
6.3.3.1.1.1 Security Protocols				
Common Internet Protocol (IP) Security Options (CIPSO) of the following emerging standard is expected to adopt MIL-STD-2045-48501, Common Security Label: Trusted Systems Interoperability Group (TSIG) Trusted Information Exchange for Restricted Environment	√			
ISP-421, Revision 1.0: The ISDN Security Program (ISP) Security Association Management Protocol (SAMP), 15 May 1994	√			
IEEE 802.10c/D13, Standard for Interoperable LAN Security-Part C: Key Management	√			

Table 16. JTA Standards (Cont'd)

Joint Technical Architecture Standards (Taken from the DoD JTA Version 1.0)	TENA Relationship			
	Facility	System Architecture	Technical Architecture	TBD
IEEE 802.10g/D7, Standard for Interoperable LAN Security - Part G: Standard for Security Labeling within Secure Data Exchange	√			x
6.3.3.1.1.2 Public Key Infrastructure Security Standards				
FIPS PUB JJJ is based on ISO/IEC 9798-3: 1993, Entity Authentication Using a Public Key System	√			
6.3.3.2 Network Security Standards				
6.3.3.2.1 Internetworking Security Standards				
RFC-1825, "Security Architecture for the Internet Protocol," R. Atkinson, August 1995.	√			
RFC-1826, "IP Authentication Header (AH)," R. Atkinson, August 1995	√			
RFC-1827, "IP Encapsulating Security Payload (ESP)," R. Atkinson, August 1995	√			
IEEE 802.10a, Standard for Interoperable LAN Security	√			

Table 17. Range Commanders Council Standards and Protocol Source Documents

Range Commanders Council Source Documents	
Document Title	**Document Number**
Underwater Acoustic Frequency Standardization	400-72
Standard Electronic Attack Clearance Request For Ranges	Inter-Range Instrumentation Group (IRIG) Standard 703-94
Video Standards and Formats	452-86
Telemetry Standards	IRIG Standard 106-96
Flight Termination Systems Commonality Standard	Standard 319-92
General Principles of Digital Filtering and a Survey of Filters in Current Range Use	155-91
IRIG Standards for Distributing Raw Radar Antenna Data	IRIG Standard 154-71
IRIG Standard for Distributing Interrange Vector Acquisition Data	IRIG Standard 152-83
Global Coordinate System	151-85
Long Haul Communications Requirements	211-92

Table 17. Range Commanders Council Standards and Protocol Source Documents (Cont'd)

Range Commanders Council Source Documents	
Document Title	**Document Number**
Frequency Standards for Radar Transponders	250-91
Parallel Binary and Parallel Binary Coded Decimal Time Code Formats	IRIG Standard 205-87
IRIG Standard Format for Interrange Exchange of Post-Mission Time-Space-Position Information	IRIG Standard 167-95
Guidelines for Interrange Graphics Capabilities	165-95
IRIG Standard Format for Global Positioning System (GPS) Data For Post-Operation Interrange Exchange	IRIG Standard 164-91
Design, Performance, and Test Standards for Flight Termination Receivers/Decoders (Volumes 1 & 2)	IRIG Standard 313-94
Coherent C-Band Transponder Standard	IRIG Standard 257-86
Noncoherent Transponder Standards	IRIG Standard 254-94
Missile Antenna Pattern Coordinate System and Data Formats	IRIG Standard 253-93
IRIG Tracking Radar Compatibility and Design Standards for G-Band (4 to 6 GHz) Radars	252-74
IRIG Standard For Pulse Repetition Frequencies and Reference Oscillator Frequency for C-Band Radars	IRIG Standard 251-80
Operations Security (OPSEC) Standards	600-87

Appendix A-Acronyms

ACETEF	Air Combat Environment Test and Evaluation Facility
AM	Asset Manager
API	Application Program Interface
ARPA	Advanced Research Projects Agency
AWT	Abstract Windowing Toolkit
BoOD	Board of Operating Directors
CDAPS	Common Data Analysis and Processing System
CTEIP	Central Test and Evaluation Improvement Program
CTTRA	Common Test and Training Range Architecture
DDM	Data Distribution Management
DII-COE	Defense Information Infrastructure-Common Operating Environment
DISA	Defense Information Systems Agency
DMS	Digital Models and Simulations
DMSO	Defense Modeling and Simulation Office
DoD	Department of Defense
DS	Distribution service
ECM	Electronic Counter Measures
ESM	Electronic Support Measures
EW	Electronic Warfare
FOM	Federation Object Model
GPS	Global Positioning System
GUI	Graphical User Interface
HPCMO	High Performance Computing and Modernization Office
HCI	Human Computer Interfaces
HITL	Hardware-in-the-Loop
HLA	High Level Architecture
HSE	Hydrophone Support Electronics
IAC	Initialization Asset Collection
ICC	Information Class Catalog
IL	Integration Laboratories
IPC	Information Presenter Class
IFF	Identify Friend or Foe
INS	Inertial Navigation System
ISTF	Installed Systems Test Facilities
IV&V	Integrated Validation and Verification
JADS	Joint Advanced Distributed Simulation
JIM	Joint Improvement and Modernization
JISTF	Joint Installed Systems Test Facility
JPO	Joint Program Office
JRRC	Joint Regional Range Complex
JSIMS	Joint Simulation System

JTA	Joint Technical Architecture
LAN	Local Area Network
LRBPM	Logical Range Business Process Model
MAC	Master Asset Catalog
MLS	Multi-Level Secure
M&S	Modeling and Simulations
MF	Measurement Facilities
NAWC-AD	Naval Air Warfare Center Aircraft Division
NBC	Nuclear Biological Chemical
NM	Network Manager
NUWC	Naval Undersea Warfare Center
OAR	Open Air Ranges
OAM&P	Operation Administration Maintenance & Provisioning
PLA	Product Line Approach
QOS	Quality Of Service
RA	Requesting Asset
RCC	Range Commanders Council
REDCAP	Real-Time Digitally Controlled Analyzer Processor
REP	Resource Enhancement Projects
RF	Radio Frequency
RIN	Range Inter-netting Networks
RTI	Run Time Infrastructure (Usually HLA RTI)
TAFIM	Technical Architecture Framework for Information Management
T&E	Test and Evaluation
TD T	Training Devices
TENA	Test and Training ENabling Architecture
TERC	Test and Evaluation Resource Council
TERIB	Test and Evaluation Reliance Investment Board
TINA	Telecommunications Information Networking Architecture
TRA	Technical Reference Architecture
TRACS	Transportable Range Augmentation and Control System
UTR	Underwater Tracking Range
VTTR	Virtual Test and Training Range
WAN	Wide Area Network

Appendix B-References

[Blaha, 1991] M. Blaha, F. Eddy, W. Lorensen, W. Premeriani, and J. Rumbaugh, "Object-Oriented Modeling and Design," Prentice Hall, Englewood Cliffs, NJ, 07632, 1991.

[Coad, 1991] P. Coad and E. Yourdon, "Object-Oriented Analysis," Yourdon Press, Englewood Cliffs, NJ, 1991.

[JTA, 1996] Joint Technical Architecture (JTA) Standards, Version 1.0, August 22, 1996 {http://www-jta.itsi.disa.mil/jta/jtacover.html}.

[RCC, 1997] Range Commanders Council Standards and Protocol Source Documents, June 1997 {http:// tecnet0.jcte.jcs.mil:9001/RCC/doculist.html

[Sun, 1997] "JAVA Development Kit," Sun Microsystems, Inc. Mountain View, Ca, 1997 {http://www.javasoft.com/products/jdk/1.1/docs/index.html}

[TAFIM, 1994] Technical Architecture Framework for Information Management (TAFIM), Version 2.0, June 30, 1994 {http://dtic.dla.mil/c3i/tafim.html}.

Appendix C-TENA Core Capabilities Details

Appendix D-Application Program Interface (API) to Distribution Services

Appendix E-TENA Object Model Diagrams